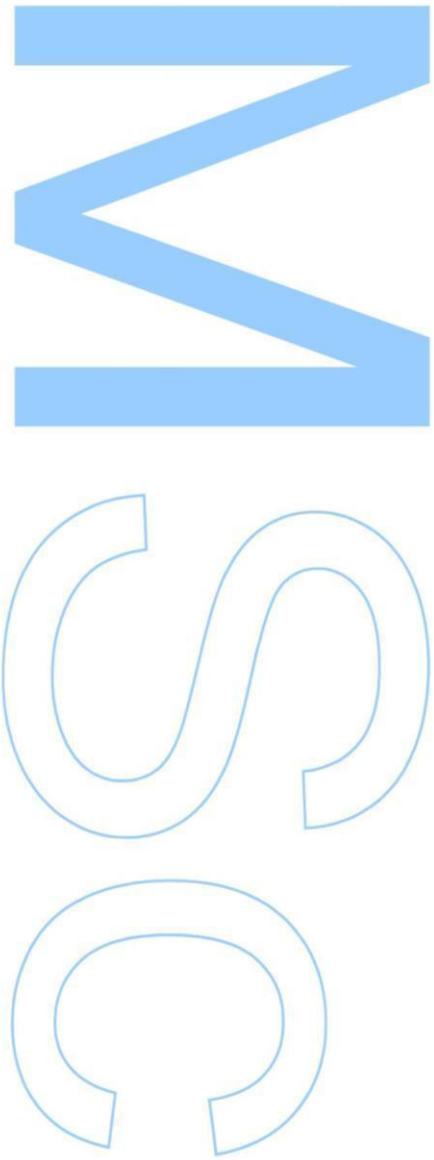
Middleware de Comunicação para Redes Oportunistas



Eder Moreno

Dissertação de Mestrado apresentada à

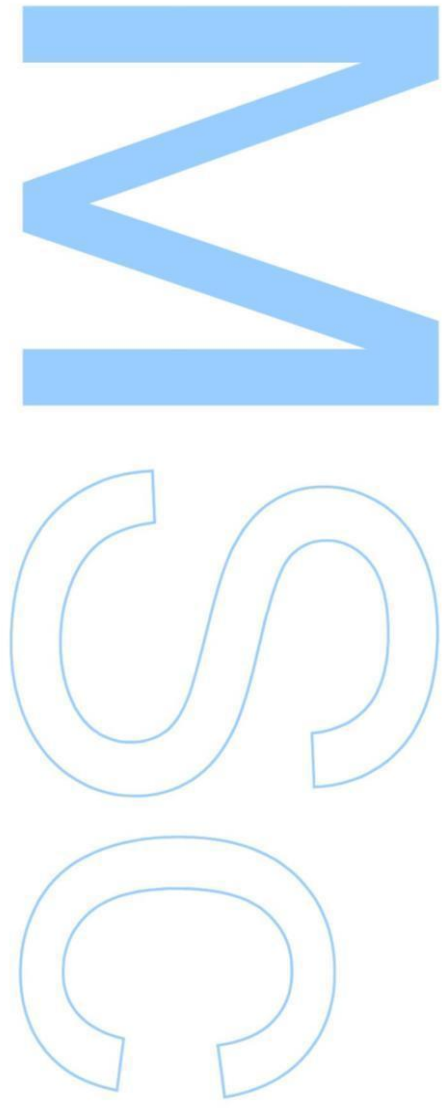
Faculdade de Ciências da Universidade do Porto em Engenharia de Redes e Sistemas Informáticos

Orientador
Professor Sérgio Crisóstomo
Professor Auxiliar,
Faculdade de Ciências da Universidade do Porto

2015

Todas as correções determinadas pelo júri, e só essas, foram efetuadas. O Presidente do Júri,

Porto, _____/_____/_____



Abstract

The ubiquity of wireless devices is generating opportunities to build networks that rely on the proximity between devices, using device-to-device communication capabilities (e.g., Bluetooth, Wi-Fi Direct) to opportunistically establish connections, aiming to convey a message to the final destination. This type of networks, known as Opportunist Networks, are an evolution of mobile ad hoc networks. They take advantage of characteristics such as high mobility of nodes and connection intermittence which are considered limitations/constraints in ad hoc networks. Opportunist Networks may become an alternative to legacy networks (e.g. cellular networks), especially in areas where there is little or no network infrastructure coverage; areas where the communication services are very expensive; or when legacy communication services become unavailable due to natural disasters or failure of power supply.

The objective of this work is to create an alternative for conventional communication networks, based on opportunistic connections between devices. We specify and implement a communication middleware called OpServiceConnection that make use of WiFi Direct and Bluetooth interfaces of Android devices, allowing them to collaborate in building opportunistic networks for communication between devices. These networks are built as an overlay topology whose underlying network are multiple Wi-Fi Direct and Bluetooth connections. Data routing between network nodes is carried out at the application layer. The underlying network topology is made of Wi-Fi Direct connections for transmission and reception of data in the same group, and uses Bluetooth connections between devices of distinct Wi-Fi Direct groups. We also develop a mechanism that allows the OpServiceConnection to automatically build its network, so the user does not have to do tasks such as searching for an available network or configure devices in order to establish connections in a network.

Resumo

A ubiquidade dos dispositivos sem fios está a gerar oportunidades de estabelecer redes entre dispositivos próximos, com capacidade de comunicação sem fios direta (e.g., *Bluetooth*, *Wi-Fi Direct*) para oportunisticamente estabelecer conexões para possibilitar a chegada de mensagens ao seu destino final. Este tipo de redes, conhecidas como redes oportunistas, são uma evolução de redes ad hoc móveis, que tiram proveito de características como a alta mobilidade dos nós e a intermitência de conexões, que são tipicamente consideradas como limitações/restrições nas redes *ad hoc*. Uma rede oportunista poderá ser uma alternativa para as redes infraestruturadas (e.g., redes celulares), principalmente em áreas onde há pouca ou nenhuma cobertura rede, áreas onde os serviços de comunicação são muito caros ou em situações de indisponibilidade dos serviços de comunicações devido a catástrofes naturais ou a falha de energia elétrica.

O objetivo desta dissertação é desenvolver uma alternativa para as redes de comunicação convencionais, baseada em comunicações oportunistas entre dispositivos. Neste trabalho, procedemos à especificação e implementação de um *middleware* de comunicação denominado de *OpServiceConnection*, que utiliza as interfaces *Wi-Fi Direct* e *Bluetooth* dos dispositivos *Android*, para permitir que colaborem entre si na construção de redes oportunistas. Estas redes são construídas como uma topologia *Overlay* que tem como camada subjacente, múltiplas conexões *Wi-Fi Direct* e *Bluetooth*. O encaminhamento de dados entre os nós da rede é realizado na camada de aplicação. A topologia de rede subjacente é construída utilizando conexões *Wi-Fi Direct* para as transmissões e recepções de dados no mesmo grupo, e conexões *Bluetooth* para as transmissões e recepções entre dispositivos de grupos diferentes. Desenvolvemos, também, um mecanismo que permite ao *OpServiceConnection* estabelecer sua redes de forma automática, em que o utilizador não terá que realizar tarefas como procurar por redes disponíveis ou configurar dispositivos.

Agradecimentos

Expresso os meus agradecimentos ao meu orientador, Professor Doutor Sérgio Armindo Lopes Crisóstomo, pelo tempo dispensado para reuniões, pelas observações, contribuições e suporte nesta dissertação.

Um agradecimento particular aos meus pais pela oportunidade, motivação para prosseguir os estudos e por tudo que têm feito nesta longa caminhada.

Aos meus avós pela criação e o apoio incondicional nesta trajetória.

A Áurea Costa, Elton Adrião, Jorge Junior e Plácido Vaz pelo apoio nesta etapa.

Por fim, agradeço a todas as pessoas que contribuíram de alguma forma para que eu possa alcançar os meus objetivos acadêmicos.

Dedico esta dissertação aos meus pais, aos meus avós, e a todas as pessoas que me acompanharam e apoiaram ao longo do meu percurso de estudos ajudando-me a atingir os meus objetivos na formação académica que se consubstancia nesta dissertação.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

1.1 Contexto e motivação

A constante diminuição dos preços e tamanhos dos dispositivos *wireless*, e o aumento acelerado das suas capacidades, permitiram um crescimento acentuado de utilizadores dos mesmos. Cada vez existem mais dispositivos de diversos tipos (como *Smartphones*, *tablets*, *smartWatchs*, etc), que incorporams tecnologias de comunicação *device-to-device* (D2D), como é o caso de *Bluetooth*, e mais recentemente o *Wi-Fi Direct*, desenvolvido pela *Wi-Fi Alliance*. Em cada lançamento, estes dispositivos trazem mais e melhor capacidade de processamento, armazenamento, autonomia da bateria, funcionalidades, que os têm tornado ubíquos.

A ubiquidade dos dispositivos *wireless* trouxe novas oportunidades e alternativas para redes de comunicação, como por exemplo, em áreas onde existe pouca ou nenhuma cobertura de infraestrutura de rede, ou em áreas onde o serviço existente é caro ou se encontra indisponível por motivos diversos (como desastres naturais)[3, 4]. Nesses cenários pode-se aproveitar a proximidade dos dispositivos para realizar, de forma oportuna, conexões *Ad Hoc* entre eles, sem recorrer a infraestruturas de redes operadoras tradicionais ou a *Access Points (AP)* para os interligar. É nesse contexto que surgem as Redes Oportunistas.

Redes Oportunistas podem ser vistas como uma evolução das redes *Ad Hoc* móveis, que aproveitam as características (como a alta mobilidade dos nós e a intermitência de conexões) tidas como limitações/constrangimentos na rede *Ad Hoc* móvel, para estabelecer conexões oportunas entre os nós, com o intuito de fazer chegar uma mensagem ao destino. Em redes Oportunistas não é suposto que os nós tenham qualquer conhecimento sobre a topologia da rede. Os nós estão habilitados a comunicar entre si, na eventualidade de surgir um caminho entre eles.

Neste trabalho é desenvolvido um sistema de comunicação alternativo às tradicionais redes infraestruturadas, onde é aproveitada a capacidade de comunicação D2D dos dispositivos *wireless*, para construir redes Oportunistas.

1.2 Objetivos

Esta dissertação tem como objetivo, desenvolver uma alternativa para as redes de comunicação convencionais baseada em comunicações oportunistas entre dispositivos. Para isso, é desenvolvido um *middleware* de comunicação que utiliza as interfaces *Wi-Fi Direct* e *Bluetooth* dos dispositivos *Android*, para permitir a colaboração entre estes na construção de redes oportunistas. Estas redes são construídas como uma topologia *Overlay* que tem como camada subjacente, múltiplas conexões *Wi-Fi Direct* e *Bluetooth*, sendo o encaminhamento de mensagens entre os nós da rede realizado na camada de aplicação. Como topologia subjacente são utilizadas conexões *Wi-Fi Direct* para as transmissões e receções de dados no mesmo grupo, e conexões *Bluetooth* para as transmissões e receções entre dispositivos de grupos diferentes.

Para além do desenvolvimento do serviço *middleware* de comunicação, é também adaptada uma aplicação para demonstrar o sistema de comunicação desenvolvido.

1.3 Estrutura da dissertação

Esta dissertação encontra-se estruturada em cinco capítulos e três apêndices. O presente capítulo descreve o contexto e a motivação do tema, bem como os objetivos propostos, e a estrutura da dissertação. No segundo capítulo, corresponde ao estado da arte do tema proposto, são descritas as tecnologias utilizadas no âmbito dessa dissertação, bem como os trabalhos relacionados.

No terceiro capítulo é especificada a aplicação desenvolvida no âmbito desta dissertação. Neste sentido descreve-se topologias *overlay* possíveis, e a topologia *overlay* utilizada na implementação da aplicação, bem como a arquitetura da aplicação. Faz-se também a especificação do funcionamento dos componentes da aplicação.

O quarto capítulo aborda essencialmente a implementação do *middleware* de comunicação proposto. Descreve também a aplicação para demonstração, abordando a adaptação da aplicação de chat *wondercom*, de forma a utilizar o serviço disponibilizado pelo *middleware* de comunicação desenvolvido.

Por último, o quinto capítulo faz uma síntese das principais conclusões, bem como propostas de trabalhos futuros.

Nos apêndices são apresentadas noções básicas sobre *Android OS* e as suas *APIs*, de forma a facilitar a compressão da implementação da aplicação. O apêndice A descreve o *Android OS*, focando essencialmente nas suas aplicações, nos seus componentes, nos processos e *Threads*, e nos meios necessários para garantir comunicações seguras entre diferentes processos/ *Threads* e entre diferentes componentes de uma aplicação. O apêndice B descreve as *API Bluetooth* do *SDK Android*. O apêndice C descreve as *APIs Wi-Fi Direct* e o protocolo de descoberta de serviço *Bonjour*, utilizado na descoberta de pares vizinhos na *framework Wi-Fi Direct*.

Capítulo 2

Estado da Arte

Neste capítulo é apresentado o estado da arte do tema proposto, começando pelas tecnologias utilizadas, onde são descritas as tecnologias de comunicação D2D (*Wi-Fi*, *Wi-Fi Direct* e *Bluetooth*) presentes nos dispositivos móveis. Segue-se a descrição de redes onde se enquadram as redes oportunistas. Ou seja, descreve-se as redes multi-hop e as suas categorias, mais concretamente redes *ad-hoc* e redes *mesh*. Em seguida descreve-se redes *overlay*, o porquê da sua utilização no contexto das limitações dos dispositivos *Android*, e por fim, descreve-se em detalhe as redes oportunistas.

Ainda neste capítulo são apresentados os trabalhos relacionados com os temas desenvolvidos nesta dissertação. Estes estão subdivididos em duas subsecções. Uma apresenta aplicações relacionadas e a outra apresenta publicações relacionadas com o trabalho desta dissertação.

2.1 Tecnologias

2.1.1 Wi-Fi

Wi-Fi é uma marca registrada da *Wi-Fi Alliance*, utilizada em produtos da classe de dispositivos *Wireless Local Area Network (WLAN)* certificados pelo mesmo, com base nos padrões da família IEEE 802.11 [5]. Atualmente *Wi-Fi* tornou-se numa das tecnologias mais comum para o acesso à Internet. Estima-se que cerca de 10% da população mundial utiliza *Wi-Fi* para se conectar à Internet [6]. Os principais padrões da família 802.11 são [6]:

- IEEE 802.11a - Opera na banda de frequência de 5GHz. Consegue transmitir a uma taxa 54Mbit/s, utilizando *Orthogonal Frequency Division Multiplexing (OFDM).*
- IEEE 802.11b - opera na banda de frequência de 2,4 GHz, utilizando a modulação *Complementary Code Keying* (CCK). Consegue atingir até 11Mbit/s de taxa de transmissão, sendo que os dispositivos adaptam-se a diferentes taxas de transmissões, nomeadamente

taxas de 5,5 Mbit/s, 2Mbit/s e 1Mbit/s de acordo com a qualidade do sinal.

- IEEE 802.11g - utiliza a mesma modulação presente no protocolo 802.11a e o acrescenta mais quatro taxas de transmissão com diferentes modulações. Consegue utilizar a velocidade de transmissão em 54 Mbit/s, mais a vantagem adicional de possuir a compatibilidade com o 802.11b.
- IEEE 802.11n - é o mais recente, que através da implementação de *Multiple-Input-Multiple-Output* (MIMO), permite aumentar a velocidade de transmissão para até 600Mbit/s. IEEE 802.11n opera tanto na frequência de 2,4 GHz como em 5GHz, o que permite ser compatível com todos os protocolos anteriores. Este também utiliza tanto canais de transmissão de 40MHz como de 20MHz, o que não acontece com os seus antecessores, que apenas o fazem em 20MHz.

2.1.2 Wi-Fi Direct

Wi-Fi Direct é uma nova tecnologia definida pela *Wi-Fi Alliance*, para suportar comunicação D2D, utilizando canais *Wi-Fi*, e não carecendo da presença de qualquer infraestrutura de rede tradicional ou *Access Point (AP)* entre os dispositivos [7]. Explora a mesma interface física para suportar comunicações sem fio padrão e D2D, definindo assim uma interface virtual para cada uma (interface Wi-Fi e Wi-Fi Direct). Suporta a taxa de transferência de dados de todos os protocolos existentes em IEEE 802.11, com a exceção do IEEE 802.11b. Tem os pontos fortes do *Wi-Fi* como performance, segurança, facilidade de utilização e ubiquidade, e acrescentou recursos que otimiza a sua utilização em cenários onde não se requer infraestrutura de rede [6].

A conexão D2D já era possível em *Wi-Fi* tradicional, através do modo de operação *Ad Hoc*. Todavia é complexo a configuração de um dispositivo para esse modo (foi designado para utilizadores avançados). Para além deste modo suportar pouca taxa de transferência de dados (a máxima é 11Mbps [8]), e os dispositivos a operar nesse modo não serem capazes de gerir simultaneamente comunicação *Ad Hoc* e infraestruturada, mantendo-as completamente separadas. De forma a ultrapassar esses constrangimentos *Wi-Fi Direct* segue uma abordagem diferente para estabelecer conexões D2D. Em vez de aproveitar o modo *ad-hoc* de operar, *Wi-Fi Direct* baseia-se no bem-sucedido modo infraestrutura do IEEE 820.11, e deixa os dispositivos negociarem entre si quem irá assumir funcionalidades equivalentes à de um *Access Point* (*"AP-Like"*). Desta forma os dispositivos *legacy Wi-Fi* poderão conectar-se facilmente em dispositivos *Wi-Fi Direct*. Essa abordagem permitiu ao *Wifi Direct* herdar todos os mecanismos de QoS (*Quality of Service*), poupança de energia, e mecanismos segurança desenvolvidos ao longo destes anos para o modo infraestrutura do *Wi-Fi* [7].

2.1.2.1 Arquitectura

Formalmente os dispositivos *WiFi Direct* são conhecidos por *P2P Devices*, sendo que a comunicação entre eles dá-se através do estabelecimento de um *P2P Group*, que funciona de uma forma

equivalente às redes de infraestrutura *Wi-Fi*. O dispositivo que desempenha as funcionalidades equivalentes a um *Access Point* no *P2P Group* é chamado de *P2P Group Owner* (P2P GO ou (GO)), e os que atuam como *Clients* são chamados de *P2P Client* [7]. Nesse sentido o GO (*P2P GO*) tem características comuns a um *Access Point* tradicionais, como:

- Anunciar a sua presença através de *beacons*, mas GO adiciona informações de elementos P2P.
- Fornecer *Dynamic Host Configuration Protocol (DHCP) server* no seu sistema, de modo atribuir endereço IP aos seus *Clients*.

P2P GO e *P2P Client* são os dois papeis possíveis que um *P2P Device* pode desempenhar dentro de um *P2P Group*, sendo que os dispositivos não conseguem realizar os dois papéis em simultâneo, a não ser que tenham mais de uma interface física ou que sejam implementadas técnicas de *time sharing* na mesma interface [7].

Dado que os papeis não são estáticos, quando dois *P2P Devices* descobrem um ao outro, negociam os seus papeis (*P2P Client* e *P2P GO*) para estabelecer *P2P Group*. Com *P2P Group* estabelecido, outros *P2P Clients* podem-se juntar ao grupo da mesma forma que em redes *Wi-Fi* tradicionais. Até mesmo os dispositivos *legacy Wi-Fi* podem se juntar ao *P2P GO*, desde que suportem os mecanismos de segurança requeridos. Nesse caso terão de suportar o WPA2PSK que é o padrão de segurança utilizado pela tecnologia *Wi-Fi Direct*. Os dispositivos *legacy Wi-Fi* conectados a um *P2P GO* são chamados de *Client Legacy*, mas não pertencem formalmente ao *P2P Group* e não têm suporte de funcionalidades definidas no *WiFi Direct*, eles simplesmente veem GO como um *Access Point* tradicional [7].

Num *P2P Group* é permitido somente aos dispositivos GO utilizar a sua interface *legacy Wi-Fi* para interligar os dispositivos do seu *P2P Group*, a uma rede externa. Também não é permitido a transferência de papel do *P2P GO* dentro do grupo, pois, se o *P2P GO* desconectar-se do grupo, o grupo é desfeito [7].

2.1.2.2 Mecanismos chave

A especificação *Wi-Fi Direct* tem definidos alguns mecanismos que considera ser chave para a sua tecnologia. Nessa dissertação iremos apenas abordar os que consideramos ser necessários para uma melhor compreensão do nosso trabalho, que são:

- *Device Discovery* - é o mecanismo que permite aos dispositivos *Wi-Fi Direct* descobrir outros dispositivos *Wi-Fi Direct* que se encontram próximo deles. Para isso é realizado um *scan* semelhante ao utilizado para detectar APs. Este mecanismo é normalmente utilizado para identificar dispositivos *Wi-Fi Direct* para se estabelecer conexões [6, 7].
- *Service Discovery* - é um recurso opcional que permite aos dispositivos *Wi-Fi Direct* anunciarem serviços suportados pelas aplicações de camada superior (por exemplo Bonjour,

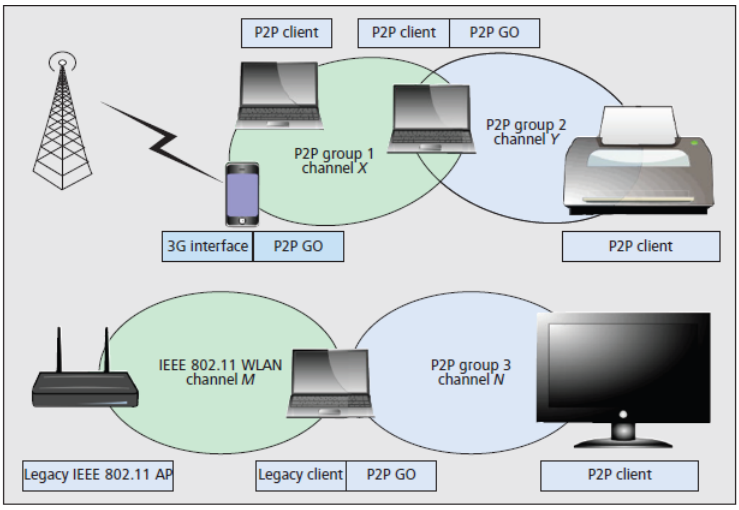


Figura 2.1: Exemplo de topologias e casos de uso suportados pelo *Wi-Fi Direct* [7]

UPnP [7]) para outros dispositivos *Wi-Fi Direct* que se encontram no seu raio de alcance [6]. A capacidade de suportar *Service Discovery* no *link Layer*, é uma característica saliente do *Wi-Fi Direct*, sendo que para o conseguir, a especificação *Wi-Fi Direct* utiliza *Generic Advertisement Service (GAS)* (especificado no padrão IEEE 802.11u), para transportar no *link Layer*, as pesquisas e respostas de descoberta de serviço geradas pelos protocolos Bonjour e UPnP [9]. Deste modo, *P2P Devices* podem realizar consultas para descobrirem o conjunto de serviços disponíveis, mesmo antes de estabelecerem um *P2P Group*, permitindo-os assim, basear no resultado da consulta obtida para decidirem se querem ou não conectar a um *P2P Device* [7] [6].

- Group Formation - Em *Wi-Fi Direct* existem dois modos de formação de grupos: *Standard* e *Autonomous*. Podendo esses ser persistente ou não. No modo *Standard*, o grupo é formado através da conexão entre dois *P2P Devices*, onde estes terão que negociar entre si quem irá realizar o papel de GO. Em *Autonomous* o grupo é formado por um *P2P Device* que se autoproclama *Group Owner* (GO) e anuncia a sua presença através de envios de mensagens *beacon*. Já no Persistent, um P2P Group pode ser declarado como persistente, durante o processo de formação do grupo pelo seu GO, que é o único que detém esse poder. Feito isso, os dispositivos que fazem parte de um *P2P Group*, irão guardar as credenciais da rede e o papel correspondente a cada dispositivo no grupo. Deste modo qualquer um dos dispositivos que pertenceu a um *P2P Group*, poderá solicitar a reconstituição desse *P2P Group* [6] [7].

2.1.3 Bluetooth

É um padrão de comunicação de baixo custo e consumo de energia, para comunicação sem fio numa curta distância [10, 11]. A tecnologia *Bluetooth* teve o início do seu desenvolvimento em 1994 pela *Ericsson Mobile Communications*, com objetivo de encontrar meios para eliminar o

excessivo número de cabos entre os dispositivos [10, 11].

Bluetooth opera na banda de frequência 2,4GHz, conhecida como banda ISM, disponível mundialmente sem a necessidade de licença. Banda ISM é uma faixa de frequência compartilhada por dispositivos, como forno micro-ondas e *Wi-Fi*. De forma a combater possíveis interferências existentes nessa banda, é utilizado a técnica de saltos de frequências (FHSS – *Frequncy Hopping Spread Spectrum*) que consiste em dividir a banda existente em canais independentes, mudando a frequência de transmissão de dados ao longo do tempo, a uma taxa de 1600 Hz [12][11].

Foram definidas pela especificação, 79 canais, mas em alguns países esse número é menor, apenas 23. Cada canal contem faixas de 1 MHz, e encontra-se dividido em intervalos de 625 micro segundos denominado *slots*, sendo que é utilizado diferentes saltos de frequência para cada *slot*. Os *slots* são utilizados alternadamente para transmitir e receber pacotes, o que resulta num esquema TDD (*Time Division Duplex*)[12][11].

Os dispositivos *Bluetooth* encontram-se classificados em 3 classes, de acordo com a potência e alcance:

- Classe 1 - com a potência máxima de transmissão de 100 mW, com um alcance de até 100 metros.
- Classe 2 – potência máxima de 2,5 mW, com um alcance de até 10 metros.
- Classe 3 – potência máxima de 1,0 mW, alcance até 1 metro.

2.1.3.1 Topologia

Em *Bluetooth* são definidas duas topologias a nível de conectividade, que são: *Piconet* e *Scatternet*. *Piconet* é a *Wireless Personal Area Network (WPAN)* formado por dispositivos *Bluetooh*, que consiste em dois ou mais dispositivos no mesmo canal, onde um desempenhará o papel de *Master* e os restantes de *Slaves*. Numa *Piconet* pode existir um único *Master*, e um máximo de 7 *Slaves* no modo ativo, e até 255 dispositivos *Slaves* não ativos. Os *slaves* só podem comunicar com o seu *Master*, num modo de comunicação ponto a ponto, não podendo comunicar diretamente com outros *Slaves* da *Piconet*. Um *Master* pode transmitir tanto ponto a ponto, para um *Slave*, como também pode transmitir ponto a multiponto, transmitindo assim para vários *Slaves* em simultâneo. A frequência de saltos no canal é baseado no endereço do *Master* de cada *Piconet*, e todos os participantes na comunicação em uma *Piconet* são sincronizados ao *clock* do *Master* [11, 12].

Scatternet são grupos de dois ou mais *Piconets* presentes ao mesmo tempo numa mesma área de atuação, com dispositivo *Bluetooth* em comum. Conexão de dois *Piconet* formam uma *Scatternet*, sendo que um dispositivo pode atuar simultaneamente em mais de uma *Piconet*, permitindo assim a possibilidade de levar a informação para além da área de cobertura de uma *Piconet*. Numa *Scatternet* um dispositivo pode atuar como *Slave* em mais de uma *Piconet*, mas

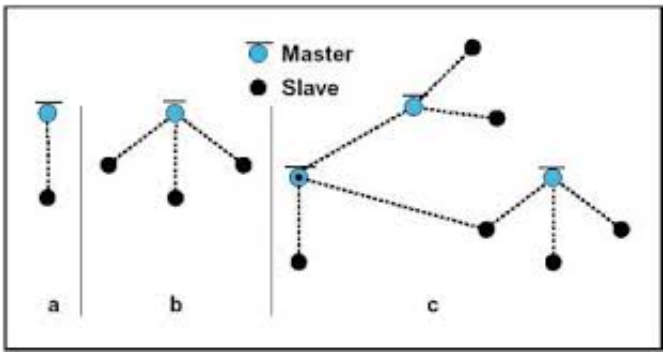


Figura 2.2: (a) *Piconet* com um único *Slave*, (b) *Piconet* com vários *Slaves*, (c) Scatternet [11]

poderá ser *Master* em apena uma delas [11, 12].

2.1.3.2 Arquitetura

A arquitetura da especificação *Bluetooth* é composta por conjunto de protocolos, *Bluetooth core*, que são os protocolos específicos da tecnologia (representados por verde claro na figura 2.3) e protocolos que não são específicos da tecnologia. Nessa dissertação serão abordados apenas os protocolos específicos da especificação *Bluetooth*, por entendermos ser os necessários para uma melhor compreensão do nosso trabalho.

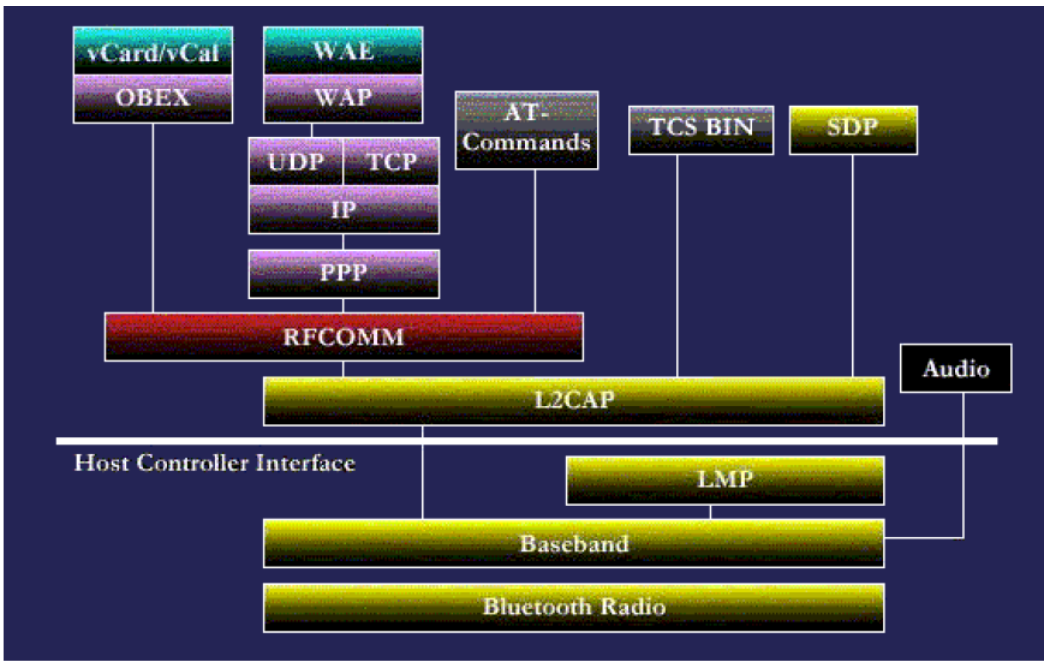


Figura 2.3: Pilha de protocolos *Bluetooth* [13]

- *Bluetooth Radio* - Esta camada é a mais baixa definida na especificação Bluetooth. Lida com a transmissão de dados via Radio Frequency (RF) e sua modulação [5].

- *Baseband* - Esta camada está acima da camada de Radio. Aqui são geridos os canais e links físicos, além de outros serviços como detecção e correção de erro, segurança. É nesta camada que é realizado os processos: *Inquiry* e *Paging*. *Inquiry* tem por objectivo realizar a descoberta de dispositivos ao alcance e recolher informações de configuração. *Paging* realiza saltos de frequência de modo a conseguir estabelecer um canal de comunicação bidirecional entre um *Master* e um *slave* [5].

  Existem dois tipos de links físicos entre as unidades *Masters* e *Slaves*, de forma a suportar aplicações multimídia que misturam voz e dados: *Synchronous Connection Oriented* (SCO) Link e *Asynchronous Connectionless* (ACL). SCO Links são conexões simétricas, comutadas por circuito (*circuit switched*), ponto-a-ponto entre um *Master* e um único *Slave* da Piconet, onde são reservados no canal dois *slots* consecutivos em intervalos fixos, um para enviar pacotes e outro para receber. Esse tipo de links são normalmente utilizados no tráfego de voz. Um *Master* pode ter até 3 links deste tipo, que podem ser empregados para a comunicação com um único *Slave* ou com *Slaves* distintos. Um *Slave* pode ter até 3 links deste tipo, caso a comunicação seja com um único *Master*, ou até 2 links, caso sejam diferentes *Masters*. ACL Links são conexões simétricas ou assimétricas, *packet-switched*, ponto-a-multiponto entre um *Master* e todos os seus *Slaves*. Esse tipo de links são tipicamente utilizados em transmissão de dados em série. Entre *Master* e um *Slave* pode existir somente um único link ACL, independentemente de haver conexões SCO estabelecidas [12][11].

- *Link Manager Protocol (LMP)* - É responsável por estabelecer e gerir um enlace entre dois dispositivos. Isto inclui desempenhar: seleção do papel (*Master* ou *Slave*), a segurança (autenticação e criptografia), controle do consumo de energia e do tamanho dos pacotes *Baseband* [5].

- L2CAP - Funciona como uma interface entre os protocolos da camada superior e a camada *Baseband*. Aqui é realizado *multiplexing* e *demultiplexing* dos dados, segmentação e remontagem de pacotes, requisitos de QoS e abstração de grupos [5].

- *Service Discovery Protocol (SDP)* - Provê meios para descobrir quais são os serviços que se encontram disponíveis nos dispositivos *Bluetooth* e quais são as suas características [5].

2.1.4 Redes Multi-hop

Redes *Multi-hop* são redes formadas por grupos de nós que comunicam entre si através de canais *wireless*, onde os nós da rede, constituem possíveis saltos no encaminhamento dos pacotes, o que possibilita múltiplos caminhos para um destino, o que pode ser útil em caso de falhas de um ou mais nós na rede [14]. Para além dos múltiplos saltos na comunicação, essas redes são também caracterizadas pelo facto de não carecerem de uma infraestrutura de rede fixa e pela forma descentralizada e auto-organizada de operar, onde os nós podem agir como router no encaminhamento de tráficos ao destino [14].

Este tipo de rede, como se pode ver na figura 2.4,tem como principais categorias de rede:

redes *Ad hoc*, redes *Mesh*, redes Sensores e redes *Hybridos* [15].

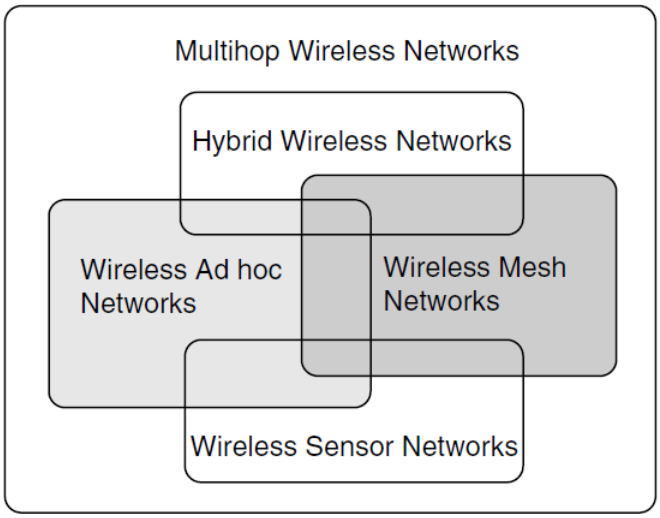


Figura 2.4: Classificação das redes *Multi-hop* [15]

Redes *Ad hoc* são redes onde os nós são terminais com capacidade de comunicação sem fio, e onde existe uma mudança dinâmica da topologia e uma alta mobilidade dos nós [15]. Redes *mesh* utilizam comunicação de múltiplos saltos em *mesh* parcial (não existe ligação direta entre todos os nós da rede), a topologia é normalmente estática, e a mobilidade dos nós é baixa, sendo que a maioria dos nós de reencaminhamento dos pacotes são fixos [15]. Redes de Sensores são formados por pequenos sensores, geralmente utilizados na coleta de informações de parâmetros físicos e transmite-as aos nós de monitoramento centrais, sendo que é possível utilizar tanto comunicação de único salto, como também se pode utilizar reencaminhamento de múltiplos saltos [15]. E por último redes *hybrid*, a qual utilizam ambas as formas de comunicação em simultâneo, a comunicação de único salto e múltiplo salto, sendo que na tradicional comunicação de único salto se encontram redes como as convencionais redes das operadoras [15].

2.1.4.1 Redes ad-hoc

Redes *ad Hoc* são conjuntos de terminais com capacidade de comunicação sem fio, que dinamicamente formam entre si redes temporárias, através de conexões *peer-to-peer*, onde alguns nós fazem parte dela apenas durante as sessões de comunicação ou enquanto estiverem numa certa proximidade dos demais nós da rede [16].

Esse tipo de rede não requere a utilização de infraestrutura fixas. Os nós são normalmente móveis podendo a qualquer momento deslocarem-se de forma independentes para quaisquer direções, afastando-se ou aproximando-se da rede, fazendo com que haja uma mudança dinâmica da topologia de rede [3][16]. Os dispositivos funcionam não apenas como terminais, mas também como router, colaborando entre si na administração da rede, e no reencaminhamento das mensagens para outros dispositivos.

Um dos exemplos mais conhecidos de rede *Ad hoc* é a Piconet formada por dispositivos *Bluetooth*, e mais recentemente *P2P Group* formada por dispositivos *wifi direct*. Estes realizam comunicações em *single hop*, assim sendo são necessários um único salto para que aconteça a transmissão da origem ao destino. Oferecem uma boa solução para cenários onde se pretende conectar dispositivos com uma distância curta entre eles, como por exemplo conectar dispositivos para troca de ficheiros, ligar por intermédio de *wireless* um ou mais computadores portáteis a uma impressora, a aparelhos de som [3]. Todavia, essas redes limitam-se a dispositivos que se encontram no mesmo raio de transmissão, neste sentido, para se ultrapassar essas limitações é necessário utilizar o paradigma *ad hoc multihop*, de forma a possibilitar comunicação em redes maiores onde existe múltiplos redes de *single hop* (*piconet*, *p2p Group*). Com isso os nós que não se encontram no mesmo raio de transmissão poderão transmitir os seus tráficos por intermédio de sequências de dispositivos intermediários [3].

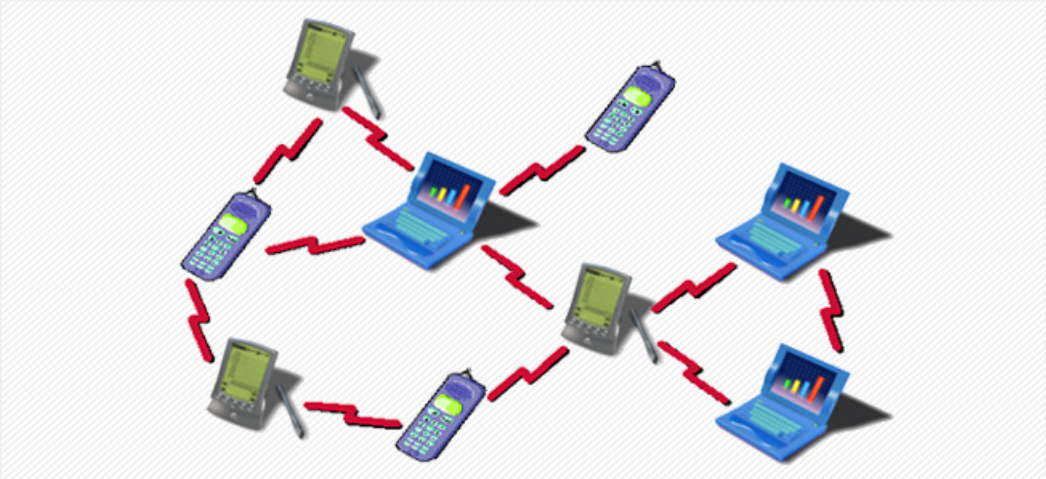


Figura 2.5: Exemplo de uma rede *Ad Hoc* [17]

2.1.4.2 Redes Mesh

Como foi mencionado na subsecção 2.1.4, rede *Mesh* a par de rede *ad hoc* fazem parte das quatro categorias de rede *multi-hop*, tendo a mobilidade dos nós e mudanças da topologia como principais diferenças entre elas, onde as redes *ad hoc* foram concebidas para ambiente de alta mobilidade dos nós, fazendo com que a topologia mude dinamicamente, e a rede *mesh* designada para um cenário estático ou mobilidade limitada, na qual maioria dos nós de retransmissão são fixos [15]. A rede *mesh* pode ser em *mesh* total em que cada nó tem ligação direta para todos os nós, ou *mesh* parcial na qual não existe ligação direta entre todos os outros nós da rede [15]. Na prática este é caracterizado por um conjunto de nós estáticos de retransmissão sem fio, proporcionando uma infraestrutura distribuída para clientes móveis em uma topologia *mesh* parcial [15].

Este tipo de rede é também caracterizado pelo seu baixo custo de implementação, manutenção, e pela tolerância a falhas, visto que utiliza o paradigma de múltiplos saltos, permitindo-lhe operar mesmo perante falhas de vários nós, desde que se mantenha pelo menos um caminho

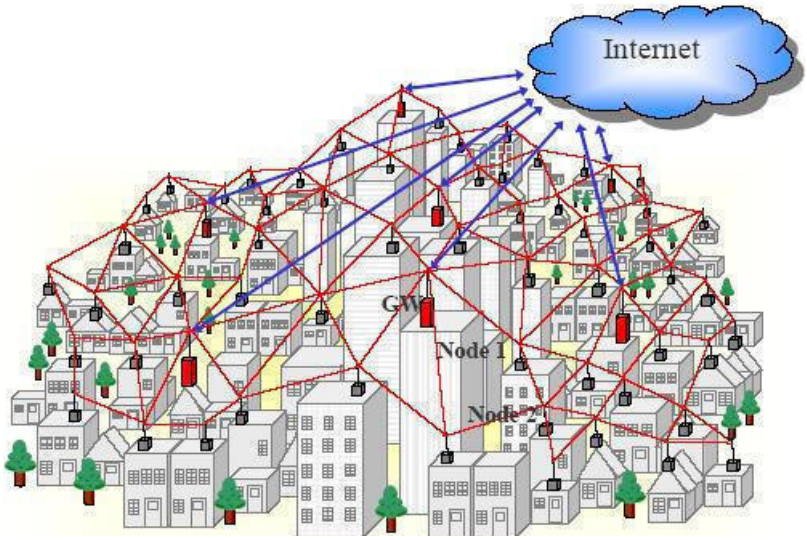


Figura 2.6: Exemplo de uma rede *Mesh* [18]

para o destino. A facilidade de configuração e escalabilidade permite-lhe ser utilizado como uma alternativa a melhorar redes já existentes em um determinado local, aumentando a área de cobertura, robustez e escalabilidade [15, 19].

2.1.5 Redes Overlay

Redes *Overlay* são redes construídas por cima de uma outra já existente, a fim de solucionar vários problemas que seriam difíceis ou impossíveis a nível da camada inferior. Atualmente a grande maioria de redes *Overlay* são construídas na camada de aplicação por cima do TCP/IP. Os nós desse tipo de redes são conectados através de links lógicos, que podem abranger várias ligações físicas. Rede *Overlay* permite introduzir funcionalidades mais complexas por cima das oferecidas pelas camadas subjacentes, permitindo assim, superar algumas das limitações subjacentes e, ao mesmo tempo, oferecer novas características de *routing* e *forwarding*, sem ter que mudar os *routers* [20].

Para o nosso objectivo, é fundamental a criação de uma camada *Overlay* sobre a redes *Ad Hoc* constituídas por dispositivos *WiFi Direct*, de forma a possibilitar a comunicação entre múltiplos *P2P Group*. Essa necessidade, muito se deve as limitações existentes por questões de segurança nos dispositivos *Android*, o que faz com que ofereçam um conjunto limitado e controlado de recursos de rede. É claro que é possível realizar *"root"* ao dispositivo, a fim de aceder recursos avançados, mas esse processo requer habilidades que estão além de um utilizador médio comum. Nesse sentido, nós optamos por não utilizar "root"nos dispositivos, mas sim, agir somente na camada de aplicação, portanto não se pode realizar nenhuma alteração no transporte ou camada de rede, ou seja, não serão realizados operações como alterar endereços IP para interfaces *Wi-Fi Direct*, configurar tabelas de roteamento [21].

2.1.6 Limitações dos dispositivos Android

Segundo a especificação *Wi-Fi Direct*, a topologia *multi-group* pode ser implementada, colocando os dispositivos móveis de forma a terem duas interfaces virtuais *Wi-Fi Direct*. Deste modo os dispositivos móveis poderão agir como ponte interligando dois *P2P Group*, através da utilização de uma interface *Wi-Fi Direct* em cada grupo. Contudo em dispositivos *Android non-rooted*, não é possível, visto que, não se consegue criar interfaces virtuais. Deste modo, apesar de não serem expressamente proibidas pela especificação, em *Android* nenhum desses cenários são possíveis: um dispositivo realizar o Papel de *P2P Client* em um grupo e *P2P GO* no outro; um dispositivo realizar o Papel *P2P GO* em mais de que um grupo; ou um dispositivo realizar o papel *P2P Client* em mais de que um grupo.

2.1.6.1 Cenários possíveis

No sistema operativo *Android* os dispositivos que suportam *Wi-Fi Direct* têm por defeito duas interfaces de rede virtuais, um de *Wi-Fi* convencional e o outro de *Wi-Fi Direct*, e podem utilizar as duas interfaces em simultâneo. Aqui os *P2P GO* dispõe de um servidor DHCP, na qual, ao ser estabelecido uma conexão *Wi-Fi Direct*, este atribui a si mesmo o endereço IP (192.168.49.1/24), e aos seus *Clients* (*P2P Client* e *Legacy Client*) (192.168.49.$\underline{x}$/24), onde $\underline{x}$ é um número aleatório entre [2;254]. De acordo com a especificação *Wi-Fi Direct* é permitido somente aos dispositivos GOs utilizar as suas interfaces *legacy Wi-Fi* para interligarem os dispositivos do seu *P2P Group*, a uma rede externa.

Nesse sentido, em *Android OS* é possível criar topologias *multi-group Wi-Fi Direct* colocando os dispositivos conectados em simultâneo como *legacy Client* de um grupo através da sua interface *Wi-Fi* , e como *P2P GO* de um outro grupo através da sua interface *Wi-Fi Direct*. A figura 3.1 representa uma topologia que reproduz este cenário. Apesar de ser possível a construção desse tipo de topologia de rede, a comunicação nestas redes carece de maior cuidados, já que algumas transferência D2D de dados entre dois GOs, ou entre um GO e os seus *Clients* não são possíveis. Isto deve-se ao facto dos GOs utilizarem as duas interfaces virtuais (*Wi-Fi* convencional e *Wi-Fi Direct*) conectados em simultâneo na rede, o que provoca alguns conflitos na comunicação realizada por esses dispositivos, devido a dois motivos:

1. O primeiro motivo é que apesar de um dos GOs agir como *Legacy Client* do outro, dois GOs vizinhos não podem comunicar diretamente entre si. Isto acontece por causa de conflito de endereços IPs. Por exemplo, na figura 3.1, pode-se ver que GO2 é *Legacy client* de GO1. Quando GO2 estiver a transmitir um pacote IP ao GO1, é colocado o endereço IP 192.168.49.1 como destino, o que implica que o pacote seja enviado ao seu lacete local em vez de ser enviado à sua interface *Wi-Fi*. Também, quando GO1 envia um pacote IP ao GO2, para o endereço IP 192.168.49.134, GO2 descarta o pacote, visto que a sua camada IP detecta que a origem do pacote corresponde ao seu endereço (192.168.49.1) [21].
2. E o segundo motivo é a ordem de entrada na tabela de roteamento dos dispositivos GOs,

implementado no sistema operativo *Android*. Por exemplo, quando um GO estiver a transmitir um pacote IP *unicast* a algum *client* do seu grupo, o pacote é sempre transmitido através da interface *Wi-Fi* do GO, uma vez que a interface *Wi-Fi* encontra-se listada na tabela de roteamento dos dispositivos *Android* com maior prioridade do que a interface *Wi-Fi Direct*. Já o envio de pacotes IP na direção contrária, de *Client* para GO é possível, visto que a tabela de roteamento do *Client* lista apenas uma interface, assim sendo não ocorre nenhum conflito [21].

2.1.7 Redes Oportunistas

Redes oportunistas podem ser vistas como evolução de redes *Ad Hoc* móveis, aproveitando-se das características como alta mobilidade dos nós e a mudança dinâmica da topologia de rede existente em redes *Ad Hoc* móveis, para se estabelecer conexões oportunas entre os nós, na tentativa de se fazer chegar a mensagem ao destino final. Portanto características tidas como constrangimento em redes *Ad Hoc* móveis, em redes oportunistas são vistas como oportunidade para possibilitar comunicação entre grupos ou nós desconectados [22].

Esse tipo de rede baseia-se no conceito do *best-effort*, aproveitando-se das oportunidades de conexão que surgem, para se fazer chegar a mensagem ao destino, gerando portanto, o caminho entre a origem e o destino de forma dinâmica, utilizando oportunamente qualquer nó como *next hop*, enquanto a mensagem não alcançar o seu destino final. Em rede oportunista não é suposto que os nós conheçam a topologia da rede, mas qualquer um dos nós pode iniciar comunicação, mesmo que não exista uma rota de comunicação. Não se assume que existe um caminho entre dois nós que desejam si comunicar, até porque os nós origem e destino podem nunca virem a estar conectados ao mesmo tempo na mesma rede. Todavia, as técnicas de redes oportunistas possibilitam a troca de mensagem entre esses tipos de nós, embora estes sejam normalmente acompanhados dos preços de atrasos adicionais na entrega das mensagens, visto que as mensagens são frequentemente colocados na rede à espera de um caminho disponível para o destino [23] [22] [24].

Redes oportunistas encontram-se normalmente subdivididas em várias regiões (sub-redes). Utiliza *Store-carry-forward* de mensagens como técnica-chave para possibilitar comunicações nas suas redes, lidando assim com a intermitência de conexão dos seus nós e permitindo a comunicação entre dispositivos de regiões (P2P Group) distintas. A técnica *store-carry-forward* de mensagens faz com que no envio de uma mensagem da origem ao destino, cada nó intermediário guarde a mensagem que recebe, carregando-a até encontrar um possível nó para a reencaminhar. Os nós intermediários irão implementar o mecanismo de comutação *store-carry-forward* das mensagens, através da criação de uma nova camada de protocolo chamado *Bundle Layer*, *Overlay* a camada de transporte [24].

Em Rede oportunista, um nó é uma entidade com *Bundle Layer*, que pode agir como um terminal, router ou *gateway*. No papel de router o *Bundle Layer* poderá guardar, carregar e reencaminhar pacotes inteiros (ou fragmentos de pacotes), entre nós dentro da mesma região. Já

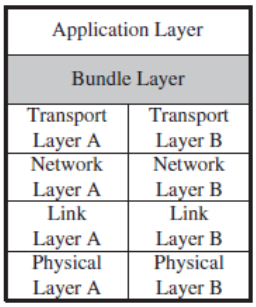


Figura 2.7: Pilha de protocolo de uma Rede Oportunista. [24]

no papel de *gateway* permite a comunicação entre diferentes regiões, possibilitando assim, que os pacotes de uma região, sejam reencaminhadas para outras regiões da rede[24].

2.2 Trabalho relacionado

2.2.1 *Publicações relacionadas*

Apresentaremos os trabalhos relacionados, mas com um foco maior em redes formadas por dispositivos *Wi-Fi Direct*, por essa tecnologia ser a base da construção de redes oportunistas adotada no nosso trabalho. Vários estudos recentes têm investigado as características e o desempenho da tecnologia *Wi-Fi Direct*, mas ainda existem poucas publicações que abordam a formação e/ou comunicação em entre vários grupos *Wi-Fi Direct*.

Em [7] é apresentado uma visão geral da tecnologia *Wi-Fi Direct* e uma avaliação experimental, utilizando dois portáteis executando Linux, onde empenham em quantificar o atraso existente na descoberta de dispositivos, a duração para associar os dispositivos, e a performance na poupança de energia.

Em [25] é proposto um algoritmo para gestão de pares para iTrust *overlay Wi-Fi Direct* que permite aos pares iniciar e manter conexões *Wi-Fi Direct*, tornando assim, a criação de grupos automatizada. iTrust é um sistema *peer-to-peer* para publicação, pesquisa e recuperação de informações, que tem por objectivo divulgar informações evitando censura.

Em [26], é criado uma arquitetura composta pelo protocolo P2PSIP sobre *Wi-Fi Direct*, de modo a proporcionar aos utilizadores dentro de um mesmo grupo, um serviço de partilha em tempo real de recursos como imagem, vídeo e endereço de *website*. O protocolo P2PSIP permite a comunicação em tempo real através da utilização do protocolo SIP de modo *peer-to-peer*. O SIP é um protocolo sinalizador da camada de aplicação, desenvolvido pelo IETF, para estabelecer, modificar e terminar sessão multimédia.

Em [8] é investigado a capacidade de criação de redes Oportunistas *overlay* à *Wi-Fi Direct framework*. Para isso são utilizados diferentes números de dispositivos *Wi-Fi Direct*, para realização de análise da performance dos protocolos em cenários reais. Deste modo apresentam o

tempo necessário para formação de grupos de diferentes tamanhos, e as configurações que melhor suportam operações de redes oportunistas e aplicações de camada superiores.

Em [27], apresenta-se uma solução para comunicação em *multi-group Wi-Fi Direct*, através de um mecanismo de *tunnelling* que irá permitir o dispositivo dentro de um *P2P Group*, transmitir dados para um outro dispositivo em *P2P Group* diferente, através de transmissões *multi-hop*, onde a comunicação é unidirecional. Segundo os autores, não existia nenhuma obra até o momento que realizaram uma avaliação experimental de comunicação em *multi-group Wi-Fi Direct*. Nesta obra os autores realizam uma avaliação experimental da arquitetura proposta por eles, utilizando dispositivos *Android "unrooted"*, onde demonstram algumas das limitações do sistema operativo *Android* investigado, e as etapas detalhada da difusão na topologia adaptada. Mais tarde, esses mesmos autores prosseguiram a investigação do tema criando um novo artigo [21], onde apresentam uma nova técnica que permite a comunicação bidirecional em *multi-group Wi-Fi Direct*, melhorando assim, a comunicação unidirecional conseguido na experiência anterior.

2.2.2 Aplicações relacionadas

A abordagem de utilizar a proximidade entre os dispositivos com capacidade de comunicação D2D para estabelecer comunicações entre os dispositivos, possibilitou o surgimento de aplicações/projetos inovadores que oferecem aos seus utilizadores a possibilidade de comunicarem entre si mesmo quando não existe conexão a Internet ou cobertura de rede celular. Nas subsecções seguintes apresentaremos algumas destas aplicações.

2.2.2.1 Briar

Briar é um software de código aberto que se encontra em desenvolvimento, com planos de longo prazo para o projeto. Este *software* destina-se a fornecer comunicação segura e resistente entre dois pontos na rede, através da implementação do conceito de rede *Mesh*. Desta forma, não utiliza servidores centralizados e a dependência de infraestrutura externa é mínima. Em Briar as conexões na rede são feitas através de *Bluetooth*, *WI-FI*, ou através da Internet, onde utilizam TOR (para impedir que os nós intermediários saibam quem está a falar com quem) e encriptação da comunicação entre a origem e o destino (*end-to-end*) [28].

2.2.2.2 Open Garden

Open Garden é um software de código proprietário, com o mesmo nome da empresa que o desenvolveu e a qual pertence. Este *software* permite aos dispositivos com capacidade de conexão sem fio (*smartphones*, portáteis, etc.) numa rede *Mesh*, partilharem acesso a Internet com outros dispositivos através da utilização de *Wi-Fi* ou *Bluetooth*. Deste modo, permite a um dispositivo que não esteja diretamente conectado á Internet disponível na rede Open Garden, realizar comunicações na Internet, por intermédio de dispositivos a qual pertencem a mesma

rede Open Garden, que tenham acesso a Internet. Esse processo é realizado de forma automática pela aplicação. Deste modo, quando o dispositivo pelo qual a conexão com a Internet está sendo compartilhado, abandona a rede, a aplicação deteta e automaticamente liga a próxima melhor conexão disponível[29].

2.2.2.3 FireChat

FireChat é também um software da empresa *Open Garden*, utiliza redes *Mesh* de forma a possibilitar a conexão *peer-to-peer* entre *smartphones* via *Bluetooth*, *Wi-Fi*, fornecendo assim, serviço de *chat* aos seus utilizadores, mesmo que estes não tenham conexão a Internet. Esta aplicação foi disponibilizada em março de 2014 uma versão para *iPhones* e 3 de abril para dispositivos *Android*, e já nesse mesmo ano tornou-se popular no Iraque, perante restrições do governo na utilização da Internet, e posteriormente em Hong Kong, durante os protestos que ali decorreu[29].

2.2.2.4 Wondercom

WonderCom é uma aplicação de código aberto, para o sistema operativo *Android*, que oferece aos seus utilizadores um serviço de chat numa WPAN formada por dispositivos *Wi-Fi Direct*. Ela permite que os dispositivos dentro de um *P2P Group*, realizem comunicação entre si, podendo os utilizadores trocar mensagens de texto, fotos, áudio, e vídeo. WonderCom não lida com a formação de grupo, trata apenas da comunicação dentro de um *P2P Group* já formado, permitindo que um dispositivo *Wi-Fi Direct* conectado enviar/receber mensagens para/de todos os dispositivos dentro do *P2P Group* a que pertencem [30].

Capítulo 3

Especificação

Neste capitulo é apresentada a especificação da aplicação Rede Oportunista Wi-Fi Direct Bluetooth (RO-WDB) desenvolvida no âmbito desta dissertação. O capítulo encontra-se subdividido em três secções: Topologias *overlay*, Arquitetura da aplicação e Especificação funcional. Na primeira secção são apresentadas algumas topologias de redes *overlay* possíveis, incluindo a topologia utilizada na implementação, e o porquê da utilização de uma das topologias *overlay* em detrimento das outras. Na segunda secção é apresentada a arquitetura da aplicação desenvolvida, onde se descreve cada um dos componentes da arquitetura e as interações existentes entre esses componentes. Por último, na secção 3, são especificados o funcionamento e a estrutura de cada um dos mecanismos a serem implementados.

3.1 Topologias overlay

As tecnologias de comunicações sem fios (*Bluetooth*, *Wi-Fi* e *Wi-Fi Direct*) existentes nos dispositivos como *Smartphones*, *Tablets*, etc, possibilitam a criação de uma série de topologias de redes *overlay*. Na presente dissertação serão apresentadas apenas as topologias que foram tidas como opção para as rede oportunistas geradas pelo *middleware* de comunicação implementado.

Em todas as figuras utilizadas para representar as topologias, optou-se por apresentar um exemplo de atribuições de endereços IP em todos os nós das topologias, de forma a facilitar a compreensão.

3.1.1 Topologia 1 - Grupos interligados pela interface *Wi-Fi* dos *GOs*

A Topologia 1 corresponde a uma topologia construída por cima de múltiplos *P2P Groups*, interligados fisicamente pelos os seus GOs, utilizando as suas interfaces *Wi-Fi* para assim, participarem como *Legacy Clients* noutros grupos.

Esta topologia encontra-se representada na figura 3.1. Os GOs são representados por círculos,

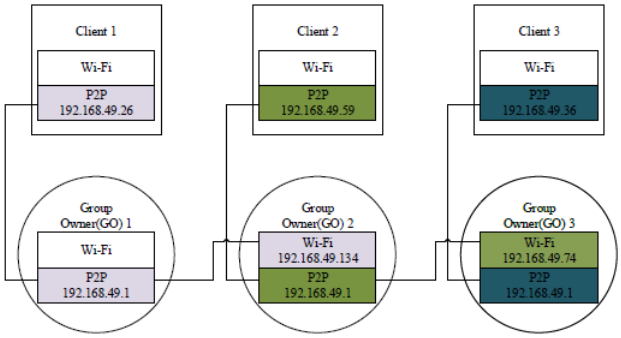


Figura 3.1: Topologia 1 - Topologia *Overlay* com dispositivos GOs a utilizar as suas interfaces *Wi-Fi* para interligarem diferentes grupos. [21]

enquanto que os *Clients* são representados por quadrados. Em todos os nós são apresentados as duas interfaces *Wi-Fi* e *Wi-Fi Direct*. As interfaces que fazem parte do mesmo *P2P Group* encontram-se destacadas com a mesma cor de fundo, com as linhas a identificarem as ligações. Neste sentido, existem seis dispositivos: três *P2P Group*; três *GOs*; e três *Clients*. É de salientar que o GO2 e o GO3 utilizam as duas interfaces (*Wi-Fi* e *Wi-Fi Direct*), agindo portanto, não só como *P2P GO* dos seus grupos, através das suas interfaces *Wi-Fi Direct*, mas também como *Legacy Client* num outro grupo através das suas interfaces *Wi-Fi*. Assim sendo, o GO2 age como *Legacy Client* do GO1, o GO3 atua como *Legacy Client* do GO2 e o GO1 não está a utilizar a sua interface *Wi-Fi*. [21].

Conforme abordado na subsecção 2.1.6, esta topologia requer alguns cuidados inerentes à sua forma de comunicação, devido a conflitos (como conflitos de endereço IP) existentes nos dispositivos *Android*, quando utilizados as duas interfaces virtuais (*Wi-Fi* e *Wi-Fi Direct*) conectadas em simultâneos numa rede. Esta necessidade de maior cuidado na comunicação, é a principal inconveniência na construção de redes com esta topologia.

A grande vantagem desta topologia quando comparada com as demais presentes nesta dissertação, traduz-se numa maior distância entre os dispositivos e numa maior taxa de transferência de dados entre os *P2P Group*, uma vez que utiliza a tecnologia *Wi-Fi* para interligar os *P2P Groups*, em detrimento de *Bluetooth*. Assim sendo, consegue usufruir da maior taxa de transferência de dados, e do maior raio de alcance, porque a Velocidade máxima de transmissão e o alcance nominal dos dispositivos *Wi-Fi* é de 54Mb/s e 100 metros respectivamente, e a dos dispositivos *Bluetooth* segundo a publicação [31] é de 1 Mb/s e 10 metros (é utilizado a classe 2 do *Bluetooth* nos *smartphones* [32]), entretanto ás versões mais recentes, apartir de 3.0 conseguem 24 Mb/s de velocidade máxima de transmissão [11].

3.1.2 Topologia 2 - Grupos interligados pelo *Bluetooth* dos *GOs*

A Topologia 2 é também uma topologia *overlay* composta por múltiplos *P2P Groups*, interligados através de conexões *Bluetooth* entre os seus *GOs*. Desta forma, consegue-se na camada da

aplicação, encaminhar mensagens de um grupo para o outro, utilizando as conexões *Bluetooth* entre os *GOs* como meio para transmissão e recepção de dados de um *P2P Group* para outro.

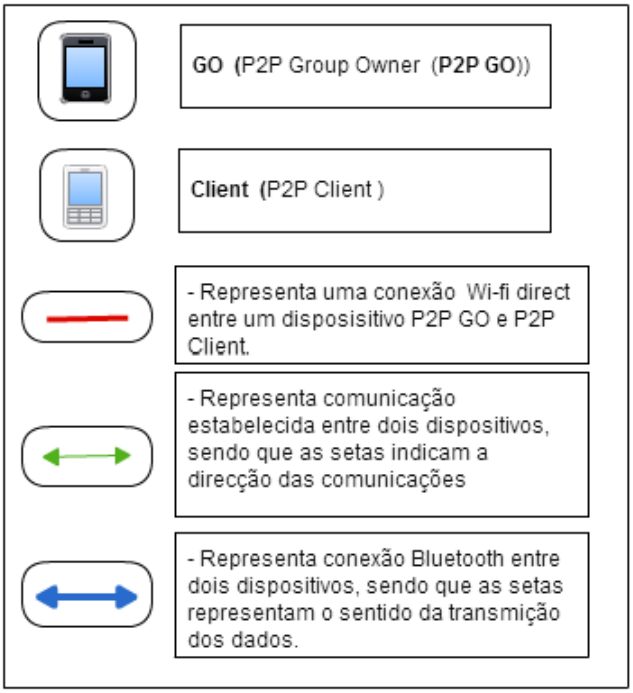


Figura 3.2: Legendas das Topologia 2 e 3

A figura 3.2 contém a legenda dos símbolos utilizados nas representações das topologias 2 e 3. Os telemóveis pretos representam *GOs*, e os brancos representam *P2P Clients*. Cada *P2P Group* encontra-se destacado através de um retângulo à volta e uma nuvem contendo o texto "Grupo", mais uma letra adicional, de forma a facilitar na distinção dos *P2P Groups*. As associações *Wi-Fi Direct* encontram-se representadas por linhas vermelhas. As linhas verdes representam as transferências diretas de dados, realizadas através dos *sockets* nas associações *Wi-Fi Direct*. Por fim, as linhas azuis representam as conexões *Bluetooth* e as transferências de dados nos seus canais.

Nesta topologia os *GOs* estão capacitados a efetivarem duas conexões *Bluetooth* em simultâneo, desempenhando o papel de *Master* em uma das conexões, e o papel de *Slave* na outra. Portanto, os GOs participam em duas *Piconets* ao mesmo tempo (*Scatternet*), o que torna o controlo das conexões *Bluetooth* mais complexo, exigindo maior cuidado para interligar dois *P2P Group* e realizar comunicações entre eles.

Esta topologia encontra-se representada na figura 3.3. Estão representados quatro *P2P Groups* interligados entre si através de conexões *Bluetooth* entre os seus *GOs*. As comunicações D2D dentro dos *P2P Groups* são realizadas apenas entre os *GOs* e os seus *P2P Clients* (comunicações bidirecionais), apesar de ser possível a comunicação D2D entre *Clients* de um mesmo *P2P Group*. É de se notar também que os *GOs* do "Grupo B" e do "Grupo C" encontram-se conectados via Bluetooth, com outros dois GO de outros grupos, realizando papel de *Master* numa das conexões e de *Slave* na outra, ou seja, interligam os seus *P2P Group* com outros dois *P2P Groups* ao

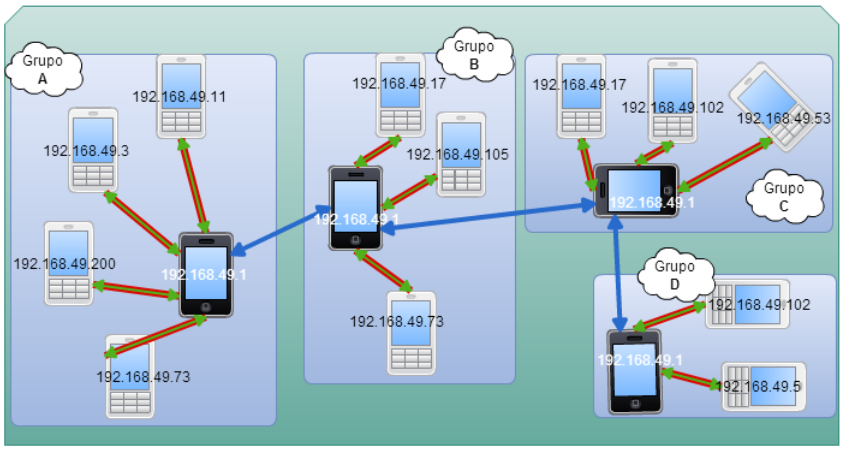


Figura 3.3: Topologia 2 - Topologia *Overlay* com apenas os dispositivos GO a utilizar *Bluetooth* para interligar diferentes grupos.

mesmo tempo. Estes dispositivos terão de lidar com o controle e encaminhamento de mensagens nos seus *P2P Groups*, como também, terão que tratar das comunicações na *scatternet*.

Esta topologia, para além de exigir um maior cuidado no controle das conexões *Bluetooth*, limita igualmente a distância entre os *P2P Groups*, uma vez que as conexões *Bluetooth* podem existir apenas entre *GOs*, o que obriga proximidade entre os *GOs*.

A grande vantagem desta topologia é o facto de exigir estratégias menos complexas para a sua construção e manutenção, e também para o encaminhamento de mensagens na rede, pois as ligações entre *P2P Groups* existem somente através dos *GOs*. Assim sendo, apenas os GOs terão a responsabilidade de interligar os seus *P2P Group* com outros, facilitando desta forma o processo de construção da topologia.

3.1.3 Topologia 3 - Grupos interligados pelo *Bluetooth* dos *GOs* e *P2P Clients*

A Topologia 3 encontra-se representada na figura 3.4. Tem as mesmas características da Topologia 2, com a diferença de utilizar apenas uma conexão *Bluetooth* por dispositivo. As ligações entre os *P2P Groups* podem ser realizadas por qualquer um dos seus nós que não estejam conectados através de *Bluetooth*. Em vez de ser apenas os *GOs* a interligar os seus grupos (como acontece na Topologia 2), nesta topologia qualquer um dos nós de um grupo (GO ou P2P Client) podem realizar conexões *Bluetooth* com qualquer nó de um outro grupo para interligarem os seus *P2P Groups*. No entanto, só poderá uma conexão *Bluetooth* por cada nó.

Com esta abordagem, a Topologia 3 consegue superar as principais limitações da Topologia 2: i) não exige tanta atenção no controle das conexões *Bluetooth*, já que permite apenas uma conexão *Bluetooth* por cada dispositivo; ii) Não depende tanto da proximidade dos dispositivos GOs para conectar múltiplos grupos, pois qualquer nó de um P2P Group (GO ou *P2P Clients*)

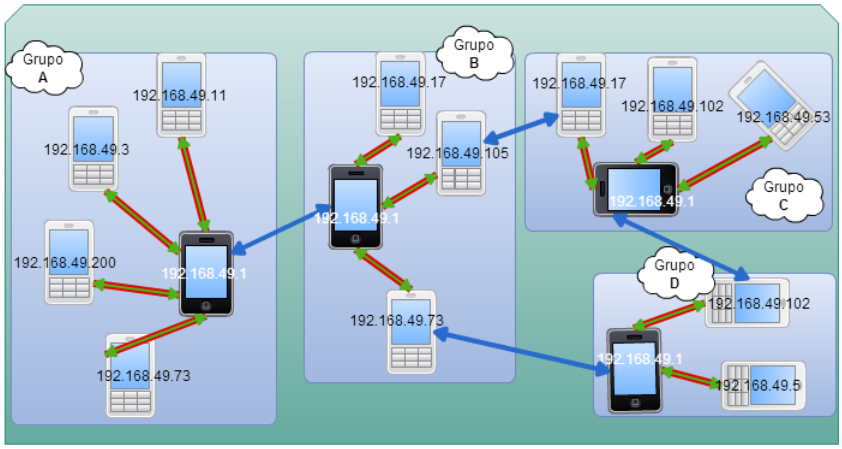


Figura 3.4: Topologia 3 - Topologia *Overlay* tendo tanto dispositivos GO como *P2P Client* a utilizar *Bluetooth* para interligar diferentes grupos

pode realizar conexões *Bluetooth* com qualquer nó de outro P2P Group, de forma a interligar os seus grupos.

3.1.4 Topologia Utilizada

Para a implementação do *middleware* de comunicação para redes Oportunistas, optamos por utilizar a Topologia 3 (figura 3.4), em detrimento das outras, considerando que esta apresenta menos limitações e menos complexidade no controle das conexões inter-grupos, e permite um encaminhamento de mensagens intra-grupo e inter-grupo mais simples.

A Topologia 3 não carece de muitos cuidados nas comunicações internas, como acontece na Topologia 1, porque nesta topologia não é preciso lidar com os conflitos nas comunicações a partir das interfaces virtuais que operam no canal *Wi-Fi* existentes nos dispositivos *Android* quando utilizado *Wi-Fi* convencional e *Wi-Fi Direct*, conectados em simultâneo. É de referir que esta topologia não exige tanta atenção no controle das conexões *Bluetooth*, não depende tanto da proximidade dos dispositivos *GOs* para interligar grupos, visto que, na Topologia 3, qualquer um dos nós de um grupo (GO e *Clients*) podem realizar conexões *Bluetooth* com qualquer nó de outro grupo de forma a interligarem os seus grupos.

Para além das vantagens apresentadas, a Topologia 3 poderá tirar partido do facto de não utilizar a interface virtual *Wi-Fi* na sua construção, utilizando-o numa possível conexão a uma rede exterior (ex: WLAN infraestruturada) .

3.2 Arquitetura da aplicação

A arquitetura da aplicação RO-WDB é composta por duas camadas: *Client* e *OpService-Connection*. *OpServiceConnection* é um serviço *Middleware* responsável por realizar todos os

procedimentos necessários para a formação de uma rede Oportunista e para a comunicação entre os seus nós, fornecendo esses recursos ao *Client*.

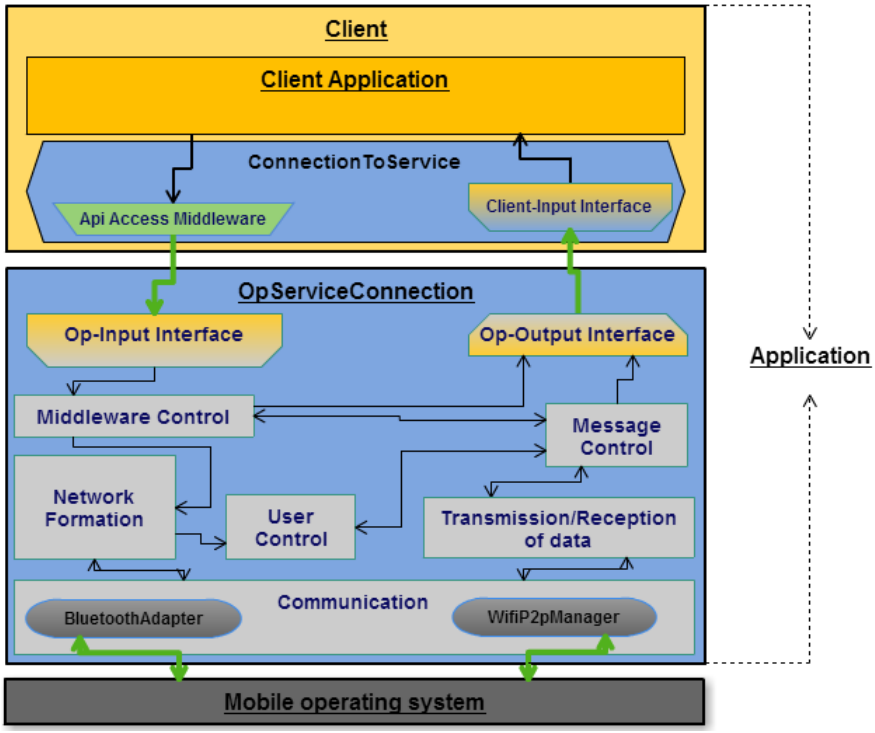


Figura 3.5: Arquitetura da Aplicação, RO-WDB

3.2.1 Client

Client é composto por duas subcamadas: o *Client Application* e o *ConnectionToService*. *Client Application* podem ser aplicações de terceiros tais como chat, jogos *multi-player*, redes sociais, ou a *wondercom* [30]. Estas aplicações usufruem dos recursos de rede oportunista fornecidos pelo *OpServiceConnection* por intermédio do *ConnectionToService*.

3.2.1.1 Client Application

Na presente dissertação utilizamos como aplicação cliente o *wondercom* [30] que oferece aos utilizadores as seguintes funcionalidades: ver mensagens enviadas e recebidas; escrever textos; gravar áudio/vídeo; tirar fotos; etc. Para o envio e recepção de mensagens entre os dispositivos na rede recorre ao componente/subcamada *ConnectionToService*.

3.2.1.2 ConnectionToService

ConnectionToService é uma subcamada responsável por intermediar as comunicações entre o *Client Application* e o *OpServiceConnection*. Para isso utiliza dois componentes: *API Access*

Middleware e *Client-Input Interface*.

- *Api Access Middleware* - é uma API que permite aceder de forma segura (trata de exceções e erros, e em alguns casos realiza também a validação dos dados de entrada) ao *Op-Intput Interface*, possibilitando a comunicação na direção do *Client Application* para o *OpServiceConnection*. Esta API permite ao *Client* inicializar os métodos que se encontram no *Op-Intput Interface* para execução de tarefas no *OpServiceConnection* (inicializar/desligar *OpServiceConnection*, enviar mensagens, etc).
- *Client-Input Interface* - é a interface que permite as comunicações na direção do *OpServiceConnection* para o *Client-Input Interface*. Através desta interface o *Client* irá receber as notificações de eventos provenientes do *OpServiceConnection*. (e.g., notificar se o *OpServiceConnection* já se encontra disponível/indisponível, *login*/*logout* do utilizador, recepção de mensagens destinadas ao *Client*).

3.2.2 OpServiceConnection

OpServiceConnection é composto por vários mecanismos que permitem fornecer os recursos da rede Oportunista para aplicações cliente. Esses mecanismos serão abordados na subsecções seguintes.

3.2.2.1 Op-Input Interface

Op-Input Interface é a interface por onde *OpServiceConnection* recebe as comunicações provenientes do *Client*. Esta interface contém diversos métodos, que podem ser acedidos pelo *Client* para solicitar ao *OpServiceConnection* a execução de uma determinada tarefa.

3.2.2.2 Middleware Control

Middleware Control tem como função intermediar os métodos existentes no *Op-Input Interface*, de forma a impedir o acesso direto dos métodos inicializados pelo *Client* aos restantes mecanismos do OpServiceConnection. Sendo assim, os métodos do *Op-Input Interface* inicializados pelo *Client*, vão solicitar ao *Middleware Control* que execute a tarefa desejada. Para além das funções acima mencionadas, este mecanismo tem também a responsabilidade de controlar o ciclo de vida do *OpServiceConnection*, controlando assim a tarefa de inicializar e desligar *OpServiceConnection*.

3.2.2.3 Network Formation

Network Formation é responsável pela construção e manutenção da topologia de rede. Este utiliza os recursos do mecanismo *Communication* (*BluetoothAdapter* e *WifiP2pManager*) para descobrir

e estabelecer conexões com os dispositivos vizinhos (Topologia 3). Conforme a arquitetura da aplicação apresentada na figura 3.5, o *Network Formation* participa nas seguintes comunicações, dentro do *OpServiceConnection*:

- Comunicação unidirecional proveniente do *Middleware Control*. Esta comunicação acontece em duas situações: Quando *OpServiceConnection* é inicializado, a comunicação é estabelecida para inicializar este mecanismo; No *shutdown* do *OpServiceConnection*, a comunicação é realizada para encerrar o *Network Formation*.

- Comunicação unidirecional deste mecanismo para User Control. Esta comunicação acontece em dois momentos: Quando o dispositivo se conecta a uma rede, o Network Formation estabelece a comunicação para inicializar o User Control; E quando o dispositivo abandona a rede, o Network Formation inicializa a comunicação para encerrar o User Control.

- Comunicação bidirecional com o mecanismo *Communication*. Esta comunicação dá-se durante todo o ciclo de vida do *Network Formation*, ou seja, acontece desde o momento que *OpServiceConnection* é inicializado até quando for desligado.

  A comunicação na direção *Network Formation* para *Communication* acontece, constantemente, em vários momentos: i) Na procura de redes *Wi-Fi Direct* disponíveis, o *Network Formation* estabelece comunicação com o mecanismo *Communication* para inicializar o serviço de descoberta de serviços existente no *Wi-Fi Direct* com intuito de encontrar dispositivos *Wi-Fi Direct* próximos com as características desejadas que possa estabelecer conexões e, por outro lado, com intuito de possibilitar a sua descoberta por outros dispositivos; ii) Para inicializar conexões *Wi-Fi Direct* com dispositivos desejados descobertos; iii) Para inicializar a descoberta de dispositivos *Bluetooth* desejados que estejam próximos para estabelecer conexões, de modo a permitir que um dispositivo seja ponte entre dois grupos *Wi-Fi Direct*, propiciando a expansão da rede.

  A comunicação no sentido inverso, do *Communication* para *Network Formation*, acontece também em diversos momentos com a finalidade de notificar *Network Formation* sobre os eventos que acontecem nos componentes que constituem o mecanismo *Communication*. Como por exemplo: Informar o *Network Formation* sobre cada dispositivo próximo encontrado, que acontece tanto pelo *WifiP2pManager* na descoberta de redes disponíveis, como também pelo *BluetoothAdapter* na descoberta de mais redes, para realizar a expansão da rede existente; Informar o *Network Formation* sempre que o dispositivo conecta ou desconecta de uma rede, que acontece tanto nas conexões *Wi-Fi Direct*, como nas conexões *Bluetooth*.

3.2.2.4 User Control

User Control é o mecanismo responsável pelo controlo dos utilizadores na rede, controlando todas as entradas e saídas de utilizadores, bem como as alterações dos seus dados durante o período que estiverem conectados à rede. *User Control* é inicializado pelo *Network Formation* logo após

a conexão do dispositivo a um *P2P Group*, e é encerrado pelo mesmo quando o dispositivo se desconecta da rede. Este mecanismo realiza comunicação bidirecional com *Message Control*, de forma a poder enviar mensagens com *OpUser* do seu próprio nó a outros nós da rede, e receber mensagens proveniente de outros nós da rede contendo os seus *OpUser*, permitindo assim, ao *User Control* de cada nó atualizar a sua lista de utilizadores na rede.

3.2.2.5 Communication

Communication utiliza as *APIs* de conectividade da plataforma *Android* (*BluetoothAdapter* e *WifiP2pManager*) para poder controlar o *Bluetooth* e o *Wi-Fi Direct*. Como se pode ver na figura 3.5, a arquitetura da aplicação, *Communication* é composto por dois componentes: o *WifiP2pManager* e o *BluetoothAdapter*. *WifiP2pManager* é responsável para aceder aos recursos do *framework Wi-Fi Direct* disponível no dispositivo móvel. E BluetoothAdapter é responsável para aceder aos recursos do *framework Bluetooth*.

3.2.2.6 Transmission/Reception of data

Transmission/Reception of data é responsável pela transmissão e recepção de mensagens de/para outros dispositivos na rede, sendo que essa transmissão e recepção de dados, pode acontecer tanto através do *Bluetooth* como através do *Wi-Fi Direct*. Este mecanismo interage com *Communication* de forma a usufruir dos recursos dos seus componentes (*WifiP2pManager* e *BluetoothAdapter*). A interação acontece tanto para enviar mensagens a outros dispositivos na rede, como para receber. Logo, existe uma comunicação bidirecional entre este mecanismo e o *Communication*.

3.2.2.7 Message Control

Message Control é o mecanismo responsável pelo controle das mensagens na rede. Controla o seu ciclo de vida, bem como os seus destinos. Este interage com vários mecanismos dentro do *OpServiceConnection*, das quais realçamos:

- *Op-Output Interface* - para poder encaminhar as mensagens destinadas ao *Client*.
- *Middleware Control* - para receber as mensagens do *Client* para as poder enviar a outros dispositivos na rede. Na direção contrária, de forma a informar ao *Middleware Control* sobre o GO que abandonou a rede (ou seja, que já não existe a rede, à qual estava conectado), para que *Middleware Control* tome as devidas providências.
- *User Control* - de modo a receber mensagens com *OpUser* do próprio nó para encaminhá-las a outros dispositivos (nós) da rede. Na direção inversa, de modo a entregar ao *User Control*, as mensagens proveniente de outros nós da rede, contendo os seus respectivos *OpUser*.
- *Transmission/Reception of data* - com vista a poder transmitir/receber mensagens para/de outros dispositivos na rede.

3.2.2.8 Op-Output Interface

Op-Output Interface é a interface por onde *OpServiceConnection* estabelece comunicações com o *Client*, enviando os dados pretendidos à porta de entrada do *Client* que é o *Client-Input Interface*. Esta interface é utilizada para notificar o *Client* sobre eventos do seu interesse que ocorrem no *OpServiceConnection*, como por exemplo:

- Quando um dispositivo recebe uma mensagem proveniente de outro dispositivo, o *Message Control* verifica se a mensagem é destinada a este, e caso seja, utiliza *Op-Output Interface* para encaminhar a mensagem ao *Client*.
- Quando *OpServiceConnection* está a ser encerrado ou já foi inicializado, o *Middleware Control* utiliza Op-Output Interface para informar ao *Client* sobre a disponibilidade/indisponibilidade do *OpServiceConnection*.

3.3 Especificação funcional

3.3.1 Construção e Manutenção da topologia

A Construção e manutenção da topologia, é da responsabilidade do componente *Network Formation* ((figura 3.5)) a sua função é construir a topologia de rede e mantê-la. Para alcançar esse objectivo, este mecanismo recorre às tecnologias *Bluetooth* e *Wi-Fi Direct* disponíveis no sistema operativo *Android*, acedidas pelo *OpServiceConnection* através dos componentes *WifiP2pManager* e *BluetoothAdapter* do mecanismo *Communication*.

Na topologia usada para a construção da rede oportunista (figura 3.4), o *Network Formation* utiliza *Wi-Fi Direct* para formar regiões (sub-redes formadas por dispositivos *Wi-Fi Direct* (*P2P Group*)) e utiliza *Bluetooth* para interligar essas regiões. Deste modo o *Network Formation* consegue expandir a rede *overlay* unindo as várias regiões na camada de aplicação.

Para que este mecanismo consiga construir e manter a topologia de forma automatizada, e mantê-la, terá que se auto configurar consoante as situações com que se depara ao longo do seu ciclo de vida. Assim sendo, durante a sua execução, ele alternará para diversos estados de execução, Conforme exemplificado na figura 3.6.

De um modo geral, *Network Formation* ao ser inicializado passa imediatamente para o estado *Discovery Wi-Fi Direct Network*, e ao terminar passa para o estado final denominado *Exit Network*. Os estados representados com fundo violeta (figura 3.6) correspondem aos estados que fazem parte do processo da construção de uma região (*P2P Group*). Vale apena realçar que nos estados *Connected P2P Client* e *Connected P2P GO* os dispositivos já se encontram conectados a um *P2P Group*, e nos estados *Discovery Wi-Fi Direct Network* e *Try Connection* os dispositivos ainda estão na fase de *discovery*/*connect*.

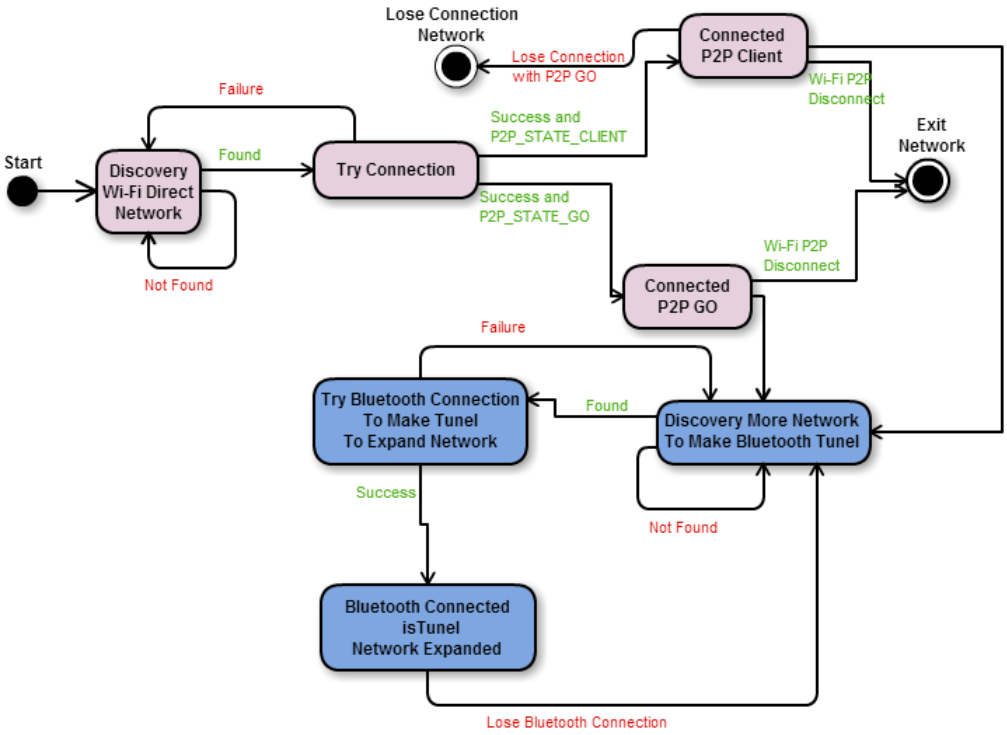


Figura 3.6: Estados de execução do mecanismo de Construção e Manutenção da topologia

Os estados representados com fundo azul correspondem ao processo de expansão da rede. Isto é, nesses estados o dispositivo já faz parte de um *P2P Group* e pretende descobrir pares que fazem parte do outro *P2P Group*, utilizando a interface *Bluetooth* para interligar os seus *P2P Group*. O primeiro estado apresentado, o *Discovery More Network To Make Bluetooth Tunel*, acontece automaticamente logo após a associação do dispositivo móvel a um *P2P Group*. O estado *Bluetooth Connected isTunel Network Expanded* corresponde a um estado em que o dispositivo já realizou expansão a rede, interligando portanto o seu *P2P Group* com outro *P2P Group* através da sua interface *Bluetooth*.

3.3.1.1 Discovery Wi-Fi Direct Network

O *Discovery Wi-Fi Direct Network* é o primeiro estado de execução do *Network Formation*. Neste estado os dispositivos móveis não se encontram conectados através do *Wi-Fi Direct*, e o *Bluetooth* ainda não é utilizado. Nesse estado são realizadas as seguintes operações:

1. O dispositivo *Wi-Fi Direct* é colocado no estado de conexão P2P_STATE_FREE. Os estados de conexão possíveis dos dispositivos *Wi-Fi Direct* na nossa rede Oportunista, encontram-se descritas na tabela 3.1.

2. O serviço de descoberta de serviços no dispositivo Wi-Fi Direct é inicializado de forma a permitir a descoberta de dispositivos *Wi-Fi Direct* próximos que tenham o serviço desejado. Na descoberta, troca-se dados tais como a informação do estado da conexão do dispositivo

Wi-Fi Direct e, no caso do dispositivo ser *P2P GO* são também passados o número de *P2P Clients*. Assim, os dispositivos, na realização da descoberta de outros dispositivos, podem realizar um filtro de modo a descobrir apenas os que se encontram nos estados desejados.

O estado *Discovery Wi-Fi Direct Network* só pode ser executado por dispositivos que se encontram no estado *P2P_STATE_FREE*. Estes, por sua vez, irão procurar apenas por dispositivos próximos que se encontrem nos estados de conexão *Wi-Fi Direct P2P_STATE_GO* ou *P2P_STATE_FREE*, sendo que os dispositivos *P2P_STATE_GO* não podem ultrapassar o número máximo de *P2P Client* estabelecido.

3. Os dispositivos bem como os seus dados (estados e número de conexões no caso de ser *P2P GO*) são guardados numa lista. Esta será acedida posteriormente, no estado *Try Connection* que irá escolher o melhor par para realizar conexões.

4. A descoberta de par é efetuada num período de tempo pré-definido (12 segundos). Caso seja encontrado algum par (ou seja, o tamanho da lista ser maior que zero), passa-se para estado *Try Connection*. Caso não seja encontrado nenhum par, mantém-se no estado *Discovery Wi-Fi Direct Network*.

Tabela 3.1: Estados de conexões dos dispositivos Wi-Fi Direct na rede Oportunista

Estados de conexões	**Descrição**
P2P_STATE_FREE	Representa dispositivos Wi-Fi Direct desconectados
P2P_STATE_GO	Representa dispositivos Wi-Direct conectados, que têm o papel de P2P GO no seu grupo
P2P_STATE_CLIENT	Representa dispositivos Wi-Direct conectados, com papel de P2P Client no seu grupo

3.3.1.2 Try Connection

O *Try Connection* acontece após o *Discovery Wi-Fi Direct Network*. Caso seja encontrado algum par para realizar conexões, o *Try Connection* acede à lista de pares encontrados e escolhe o melhor par para tentar se conectar. Caso não consiga, escolhe o segundo melhor e tenta novamente, e assim sucessivamente.

Definimos como melhor par, os dispositivos que se encontrem no estado *P2P_STATE_GO* e que possuam o menor número possível de *P2P Clients*. Contudo, a escolha dos dispositivos *P2P_STATE_FREE* como melhor par é feita apenas no caso de não haver dispositivos *P2P_STATE_GO* na lista.

Entre os dispositivos *P2P_STATE_FREE*, é escolhido aleatoriamente, o melhor par para se estabelecer conexão. No caso de se tentar a conexão com todos os pares da lista, e não se bem

for sucedido, volta-se novamente para o estado *Discovery Wi-Fi Direct Network*. Se tiver sucesso, verifica-se uma mudança de estado dependendo do papel que o dispositivo irá desempenhar no seu grupo:

- caso desempenhe o papel de *P2P GO*, será mudado para o estado *Connected P2P GO*.
- caso desempenhe o papel de *P2P Client*, será mudado para o estado *Connected P2P Client*.

3.3.1.3 Connected P2P Client

O *Connected P2P Client* é um estado em que o dispositivo já está associado à rede Oportunista e encontra-se num *P2P Group* desempenhando o papel de *P2P Client*. Nesta situação o *Network Formation* transita para o estado *Discovery More Network To Make Bluetooth Tunel*, de forma a inicializar o processo de expansão da rede. A partir deste estado pode-se mudar para os seguintes estados:

- *Lose Connection Network* - em caso de perder a conexão com o seu *P2P GO*, por motivos de abandono da rede por parte do *P2P GO*, ou porque o dispositivo se afastou do raio de alcance do seu *P2P GO*.
- *Exit Network* - Se o nó abandonar a rede.

3.3.1.4 Connected P2P GO

O *Connected P2P GO* é também um estado em que o dispositivo já está associado à rede Oportunista, mas desempenha o papel de *P2P GO* no seu *P2P Group* (região). Neste situação, o *Network Formation* transita para o estado *Discovery More Network To Make Bluetooth Tunel*, da mesma forma e com o mesmo objectivo do estado *Connected P2P Client*, que é inicializar o processo de expansão da rede. A partir desse estado pode-se também mudar para o estado *Exit Network*, se *Wi-Fi Direct* for desconectado.

3.3.1.5 Discovery More Network To Make Bluetooth Tunel

O *Discovery More Network To Make Bluetooth Tunel* é definido como um estado em que o dispositivo já se encontra conectado a um *P2P Group* da rede Oportunista, inicializando a descoberta de pares através do *framework Bluetooth*, procurando por pares que fazem parte de outros *P2P Group* e que não tenham nenhuma conexão a nível de *Bluetooth*, a fim de estabelecer conexão *Bluetooth* com um desses pares encontrados, e consequentemente interligarem os seus *P2P Group*, expandindo assim, a rede na qual fazem parte.

Este processo decorre num período de tempo de 12 segundos (é o tempo que demora o processo de descoberta de pares nos dispositivos *Bluetooth*), durante o qual os dispositivos desejados serão

colocados numa lista. Após este processo, caso não seja encontrado nenhum par (ou seja, a lista estiver vazia), o processo de descoberta de pares é reinicializado. Caso contrário, muda-se para o estado *Try Bluetooth Connection To Make Tunel To Expand Network* para tentar conexões com os dispositivos encontrados, visando realizar uma expansão da rede.

Tabela 3.2: Estados de conexões dos dispositivos Bluetooth na rede Oportunista

Estados de conexões Bluetooth	**Descrição**
BT_FREE	Representa um dispositivo que não esteja conectado a nível de Bluetooth
BT_TUNNEL	Representa um dispositivo que esteja conectado a nível de Bluetooth, ou seja, é uma ponte entre dois P2P Group

3.3.1.6 Try Bluetooth Connection To Make Tunel To Expand Network

No estado *Try Bluetooth Connection To Make Tunel To Expand Network* será escolhido, de forma aleatória um dispositivo na lista dos dispositivos encontrados na descoberta de pares realizado no estado anterior, a fim de estabelecer uma conexão *Bluetooth*, podendo ocorrer os seguintes casos:

- Não se conseguir realizar conexões com nenhum dos pares da lista. Regressa-se ao estado *Discovery More Network To Make Bluetooth Tunel*.
- Conseguir realizar conexões com algum dos pares da lista. Muda-se para o estado *Bluetooth Connected isTunel Network Expanded*.

3.3.1.7 Bluetooth Connected isTunel Network Expanded

No estado Bluetooth Connected isTunel Network Expanded, o dispositivo móvel já deu o seu contributo no processo de construção da topologia de rede, ou seja, para além do dispositivo móvel estar conectado a um *P2P Group*, este assume igualmente o papel de ponte interligando o seu *P2P Group* com um outro *P2P Group*. Posto isto, nesse estado a rede Oportunista já tem mais do que uma região (*P2P Group*) e no caso de se perder a conexão *Bluetooth* (perder a ligação com outro *P2P Group*), este regressa novamente ao estado *Discovery More Network To Make Bluetooth Tunel*.

3.3.1.8 Lose Connection Network

O *Lose Connection Network* é um estado final de *Network Formation*. Acontece quando um dispositivo *Wi-Fi Direct* conectado num *P2P Group*, desempenhando o papel de *P2P Client*, perde a conexão com o seu *P2P GO*. Contudo, antes de ser desligado notifica-se o *Client*, de

que perdeu conexão à rede e terá que inicializar o *OpServiceConnection*, se pretender conectar novamente à rede.

3.3.1.9 Exit Network

O *Exit Network* é, também, um estado final do *Network Formation*. Acontece quando o *OpServiceConnection* se encontra no processo de *shutdown*, normalmente executado pelo *Middleware Control*. Deste modo *Network Formation* irá também terminar o seu ciclo de vida, desconectando as interfaces conectadas (*Wi-Fi Direct* e *Bluetooth*), abandonando qualquer rede que esteja ligada.

3.3.2 Especificação de mensagens

A especificação de mensagens (figura 3.7) baseia-se numa estrutura denominada *OpPackage* que foi adaptada apartir da estrutura *Message* do *wondercom [30]*. O OpPackage é composto por 15 campos, sendo os campos *id*, *Type*, *isMine*, *ChatName*, *text*, *senderAddress*, *receiveAddress*, *fromAddress* e *toAddress* obrigatórios, e os restantes opcionais, dependendo do tipo de mensagens (Tabela 3.3).

A tabela 3.4 contém a descrição de todos os campos da estrutura *OpPackage*. De modo a facilitar compreensão de *OpPackage*, utilizamos diferentes tipos de cores para distinguir as diferentes categorias de funcionalidades dos campos das mensagens representadas na figura 3.7.

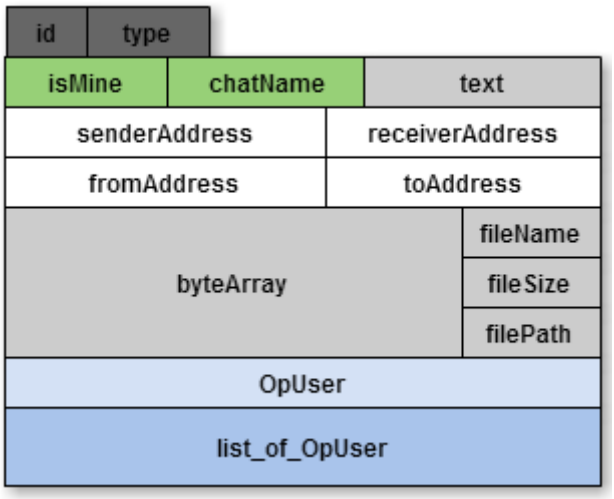


Figura 3.7: Estrutura da mensagem (OpPackage).

Os campos *id* e *type* (com fundo cinzento escuro) permitem distinguir as mensagens na rede. O id contém a identificação única de um *OpPackage* de um utilizador. E o *type* contém o tipo de mensagem (*OpPackage*) a ser utilizado, podendo ser qualquer um dos tipos da tabela 3.3.

Os campos Campos *isMine* e *chatName* (fundo verde) são utilizados na identificação das

Tabela 3.3: Tabela com a lista de tipos OpPackage, e suas respectivas descrições

Tipo de Mensagens	**Descrição**
TEXT_PACKAGE	Mensagem de texto simples.
IMAGE_PACKAGE	Mensagem de imagem
VIDEO_PACKAGE	Mensagem de vídeo
AUDIO_PACKAGE	Mensagem de áudio
FILE_PACKAGE	Mensagem de diferentes tipos de ficheiros
USER_CONNECTION_PACKAGE	Mensagem contendo OpUser. É utilizado no controlo de utilizadores na rede.
USERS_CONNECTION_PACKAGE	Mensagem contendo uma lista de OpUser. É utilizado também no controlo de utilizadores na rede.

mensagens, no *Client Application*. Através do campo *isMine*, o *Client Application* consegue distinguir para os seus utilizadores, as mensagens geradas por eles e as que receberam. O *chatName* contém o nome que está a ser utilizado pelo utilizador dono da mensagem (ou seja, utilizador onde foi gerada a mensagem).

No que diz respeito aos campos *senderAddress*, *receiverAddress*, *fromAddress* e *toAddress* (fundo a branco) são utilizados na identificação do caminho de cada mensagem. O *senderAddress* e *receiverAddress*, representam o último salto mensagem, identificando quem enviou (*senderAddress*) e quem receberá a mensagem. Já o *fromAddress* e o *toAddress* representam a origem inicial e destino final da mensagem.

Os campos opcionais *text*, *byteArray*, *fileName*, *fileSize* e *filePath* (fundo cinzento claro) contém os dados que se deseja transmitir. O campo *text* contém textos simples (*String*) que se pretende transmitir. O *byteArray* contém os *bytes* dos dados mais complexos, como ficheiros de áudio, vídeo, etc. O campo *byteArray* vem sempre acompanhado dos campos *fileName*, *fileSize* e *filePath*, que correspondem ao nome, tamanho e endereço físico no dispositivo do ficheiro a ser transmitido no *byteArray*. Nesta linha, estes apenas serão utilizados quando o tipo de dados (descritos na tabela 3.3) utilizado for: *TEXT_PACKAGE*, *IMAGE_PACKAGE*, *VIDEO_PACKAGE*, *AUDIO_PACKAGE*, ou *FILE_PACKAGE*.

Por último, *OpUser* e *List_of_OpUser* (fundo azul) representam os dados dos utilizadores a serem transmitidos. São utilizados pelo *User Control* no controlo de utilizadores na rede. Esses campos, assim como o *byteArray*, são também utilizados dependendo do tipo de mensagens utilizado (Tabela 3.3). *OpUser* contém dados de um único utilizador. É utilizado apenas quando o tipo de mensagem for *USER_CONNECTION_PACKAGE*. Enquanto que o *List_of_OpUser* que contém uma lista de utilizadores (*OpUser*) de um *P2P Group*, é utilizado somente quando o tipo de mensagem for *USERS_CONNECTION_PACKAGE*.

Tabela 3.4: Descrição dos campos que constituem OpPackage

Campo	**Descrição**
id	Contém um identificador único de uma mensagem (OpPackage) de um dispositivo da rede. É obrigatório numa OpPackage, e é fundamental na realização do controle das mensagens na rede.
type	É um campo obrigatório em todas as mensagens (OpPackage), é responsável por definir que tipo de mensagem será OpPackage. Os tipos de mensagens possíveis encontram-se na tabela 3.3.
isMine	É um campo utilizado por Client Application, de forma a diferenciar para os seus utilizadores, se a mensagem foi gerada por eles, ou oriundo de outros utilizadores da rede.
chatName	Este é também utilizado por Client Application. Contém o nome que o utilizador está a utilizar na rede.
text	Este contém um texto simples, a qual se pretende transmitir.
senderAddress	Contém o endereço (campo id do OpUser) do nó de onde partiu a transmissão da mensagem, ou seja, o endereço do nó de último salto da mensagem.
receiverAddress	Contém o endereço (id do OpUser) do nó que receberá a mensagem transmitida.
fromAddress	Contém o endereço (id do OpUser) do nó onde foi gerada a mensagem, ou seja, o endereço origem da mensagem.
toAddress	Contém o endereço (id do OpUser) do nó que será o destino final da mensagem.
byteArray	Este campo contém os bytes dos ficheiros que se pretende transmitir. É utilizado ou não, dependendo do tipo de mensagem utilizada, podendo ser: TEXT_PACKAGE, IMAGE_PACKAGE, VIDEO_PACKAGE, AUDIO_PACKAGE„ou FILE_PACKAGE. Cada byteArray é utilizado em conjunto com fileName; fileSize e filePath.
fileName	Contém o nome do ficheiro a ser transmitido no byteArray.
fileSize	Contém o tamanho do byteArray.
filePath	Contém a localização física do ficheiro dentro do dispositivo.
OpUser	Contém OpUser 3.8 a ser transmitido. É utilizado somente quando USER_CONNECTION_PACKAGE for o tipo de mensagem utilizado.
List_of_OpUser	Contém uma lista de OpUser a ser transmitido. É utilizado somente quando USERS_CONNECTION_PACKAGE, for o tipo de mensagem utilizado.

3.3.2.1 Estrutura de OpUser

O *OpUser* é uma estrutura de dados que representa um nó da rede Oportunista, contendo na sua estrutura onze campos, que correspondem aos dados do utilizador e aos dados das interfaces de comunicação do dispositivo móvel. A estrutura do *OpUser* encontra-se representado na figura 3.8, e os seus campos encontram-se descritos detalhadamente na tabela 3.5.

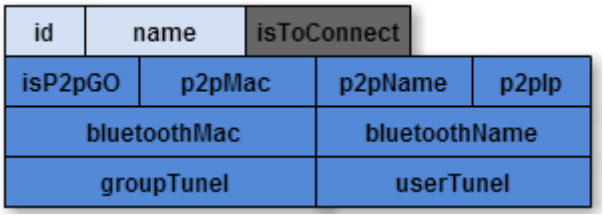


Figura 3.8: Estrutura de OpUser

Os dados do utilizador são os campos *id* e *name*, representados na figura 3.8, com fundo azul claro. O *id* é designado para identificar, de forma única, um utilizador na rede. O *name* contém o nome utilizado pelo utilizador. Já os dados das interfaces de comunicação do dispositivo móvel, nomeadamente a interface *Bluetooth* e a interface *Wi-Fi Direct*, encontram-se representados na figura 3.8, com fundo azul escuro, onde a primeira linha contém os dados relativos à interface *Wi-Fi Direct* (*isP2PGO*, *p2pMac*, *p2pName* e *p2pIp*) e a segunda os dados da interface *Bluetooth* (*bluetoothMac* e *bluetoothName*).

Por fim, a última linha é composta pelos campos *userTunnel* e *groupTunnel*. O *userTunnel* corresponde ao identificador único (*id* do *OpUser*) do par do outro *P2P Group*, no qual é estabelecido conexão *Bluetooth* para interligar o seu *P2P Group* com a do dispositivo em questão. O *groupTunnel* corresponde ao identificador único (*id* do *OpUser*) do nó que é *P2P GO* do par, a qual o seu id é transportado no campo *userTunel*.

3.3.3 Mecanismo de controlo de utilizadores

O mecanismo de controlo de utilizadores é representado pelo *User Control*, conforme a arquitetura da aplicação (figura 3.5). Tem como principal objectivo controlar todos os utilizadores na rede. Por conseguinte, controlará todas as entradas e as saídas de utilizadores na rede, guardando cada utilizador que entra na rede numa lista, e removendo da lista os utilizadores que abandonam a rede.

Cada nó (OpUser) na rede é um objecto contendo as informações do utilizador, além dos dados adicionais do seu dispositivo, que são indispensáveis no encaminhamento das mensagens na rede.

Para que este mecanismo consiga desempenhar as suas funções, utilizamos dois níveis de controlo de utilizadores. O primeiro nível consiste em controlar utilizadores intra-grupo (*P2P Group*) e o segundo nível consiste em controlar os utilizadores na rede em geral, cujo os utilizadores

Tabela 3.5: Descrição dos campos que constituem OpUser

Campos	**Descrição**
id	Contém um identificador único de utilizador (OpUser) na rede.
name	Contém o nome do utilizador.
bluetoothMac	Contém o endereço MAC da interface Bluetooth do dispositivo móvel.
bluetoothName	Contém o nome da interface Bluetooth do dispositivo móvel.
p2pMac	Contém o endereço MAC da interface Wi-Fi Direct do dispositivo.
p2pName	Contém o nome da interface Wi-Fi Direct do dispositivo móvel.
p2pIp	Contém o endereço IP atribuído a interface Wi-Fi Direct do dispositivo móvel na rede (P2P Group).
isP2pGO	Este componente identifica se um dispositivo móvel é P2P GO no seu grupo ou não. isP2pGO contém true no caso de ser P2P GO, e false, caso contrário.
isToConnect	Este componente define se OpUser está a entrar na rede, ou se está a abandoná-la.
userTunnel	Este componente é utilizado para identificar se o nó realiza o papel de ponte entre o seu P2P Group e um outro, ou não. No caso de ser ponte, userTunel conterá o identificador único do nó (id do OpUser), o qual o dispositivo realiza conexões Bluetooth, de modo a estabelecer interligação entre os dois P2P Group. E se não for ponte, userTunnel é null.
groupTunnel	Este componente é diferente de null, caso o nó realize o papel de ponte entre o seu P2P Group e um outro. Com isso, groupTunnel conterá o identificador único do nó (id do OpUser), que é P2P GO do nó, o qual é realizado conexões Bluetooth para se estabelecer a ponte entre os dois P2P Group.

são controlados em múltiplos *P2P Groups*, ou controlo inter-grupos.

3.3.3.1 Controlo de utilizadores intra-grupo

O controlo de utilizadores dentro de um grupo ocorre logo após a inicialização do *User control*, ou seja, após a conexão de um dispositivo móvel num *P2P Group*. Neste sentido, em cada dispositivo móvel, o *User Control* prepara o seu *OpUser* e inicializa a sua lista de utilizadores na rede.

Em caso do dispositivo desempenhar o papel de GO no seu grupo, o *User Control* enviará uma mensagem com o seu *OpUser* (contendo o campo *isToConnect* a *true*), para todos os dispositivos *P2P Client* no ato da conexão, e vice versa.

A cada mensagem contendo *OpUser* dos seus *P2P Client* recebidas no *User Control*, este irá atualizar a sua lista de utilizadores, adicionando ou atualizando caso *isToConnect* for *true*, e

removendo caso *false*. Em seguida irá reencaminhar a mensagem aos restantes dispositivos *P2P Client* do seu grupo. No caso de haver alguma alteração no *OpUser* de um dispositivo, o seu *User Control* reenviará uma mensagem com o seu *OpUser* alterado, a todos os *P2P Client* da sua lista. E no caso do dispositivo estar a desconectar da rede, o seu *User Control* enviará uma mensagem contendo o seu *OpUser*, a todos os seus *P2P Clients*, mas contendo *isToConnect* a *false*.

No caso do dispositivo desempenhar o papel de *P2P Client* no seu grupo, *User Control* irá enviar uma mensagem com o seu *OpUser* para o dispositivo *P2P GO* do seu grupo, colocando o campo *isToConnect* a *true*, caso o utilizador esteja a entrar na rede ou a alterar os seus dados, e colocando o campo *isToConnect* a *false*, caso o utilizador estiver a abandonar a rede. Igualmente, *User Control* irá guardar o *OpUser* do seu GO numa variável.

3.3.3.2 Controlo inter-grupos de utilizadores

O controlo inter-grupos de utilizadores (o segundo nível de controlo de utilizadores), entra em funcionamento a partir do momento em que é realizado algum tipo de expansão a rede, isto é, quando ocorre alguma conexão *Bluetooth* entre dispositivos de diferentes *P2P Group*. A técnica utilizada pelo *User Control* para esse fim é a seguinte:

1. Quando dois dispositivos de diferentes *P2P Group* estabelecem conexão *Bluetooth* entre si, os seus *User Control* enviarão entre si duas mensagens, uma contendo os seus respectivos *OpUser*, e a outra contendo OpUser dos seus *P2P GO*.

2. Nos dispositivos que recebem as mensagens com *OpUser* através do *framework Bluetooth*, o *User Control* guardará os dois *OpUser* numa variável, e atualizará os seus próprios *OpUser*, preenchendo os campos *userTunnel* e *groupTunnel*. O *userTunnel* é preenchido com o *id* do *OpUser* do dispositivo, o qual é estabelecido na conexão *Bluetooth*. O *groupTunnel* é preenchido com o *id* do OpUser do dispositivo *P2P GO* com o qual foi estabelecida a conexão *Bluetooth*.

3. Quando dois dispositivos conectados entre si a nível de *Bluetooth* se desconectam um do outro, esses removem os *OpUser* das variáveis e atualizam os seus próprios *OpUser*, colocando os campos *userTunnel* e *groupTunnel* a *null*.

3.3.4 Encaminhamento e controlo de mensagens

O mecanismo de encaminhamento e controlo de mensagens (representado por *Message Control* (figura 3.5)) tem como principal objectivo o controlo das mensagens na rede, bem como o seu encaminhamento. Portanto, todas as mensagens recebidas/enviadas por um dispositivo da rede, passam por esse mecanismo que decide desta forma os seus destinos. *Message Control*, baseia-se nos dados que se encontram nos campos de cada *OpPackage* (*id*, *type*, *senderAddress*, *receiverAddress*, *fromAddres* e *toAddres*), a fim de decidir que destino deve dar a cada mensagem

que passa por ele. Assim, este decide se deve entregar uma mensagem ao *User Control* ou encaminhá-la ao *Client*/outros dispositivos na rede.

No encaminhamento de mensagens optamos por utilizar *broadcast* de mensagens para todos os nós da rede. Assim, cada mensagem gerada e enviada por um nó na rede, é encaminha a todos os outros nós conectados na rede. Para tal utilizamos as seguintes técnicas no encaminhamento das mensagens:

1. Um nó não pode reencaminhar uma mensagem para o nó que o enviou.
2. Os nós *P2P Client* enviam mensagens apenas aos seus respectivos *GOs* que irão tratar de reencaminhar para os restantes *P2P Client* dos seus *P2P Group*.
3. Os *GOs* enviam as mensagens para todos os seus *P2P Clients*.
4. Para encaminhar mensagens a um outro *P2P Group* da rede, são utilizados os nós que realizam papel de ponte para um outro *P2P Group* para as encaminhar, ou seja, a comunicação inter *P2P Group* pode ser realizada apenas através dos nós "Ponte", portanto através de *Bluetooth*.

Para um maior controlo de mensagens na rede, assim como uma melhor utilização das técnicas acima referidas adotamos as seguintes estratégias:

- Cada nó na rede guarda uma cópia das mensagens antes de as enviar. Isto faz com que os nós ao receberem uma mensagem verificam se já não tem a mensagem na sua lista. Caso tenha *Message Control* descarta a mensagem e caso contrário este guarda-a na sua lista por um certo período de tempo.
- Para identificar uma mensagem, o *Message Control* verifica os campos *id* e *fromAddres*. Esses dois campos juntos correspondem a identificação única de uma mensagem na rede, uma vez que *id* contém um identificador único das mensagens geradas por um dispositivo, e *fromAddres* contém o identificador único (*id* do *OpUser*) do nó (dispositivo) que gerou a mensagem.
- cada mensagem guardada na lista de mensagem permanece na lista até um certo período de tempo estabelecido pelo *Message Control*. Para alcançar essa estratégia, guardamos numa lista chave valor (HashMap) o *DateTime* que foi guardada a mensagem como valor, e o *id* da mensagem como a chave, em paralelo com as mensagens guardas. Ou seja, para cada mensagem guardada na lista de mensagens, é guardado em simultâneo no *HashMap* o *DateTime* que a mensagem foi guardada. Neste sentido, sabendo os momentos (*DateTime*) que as mensagens foram guardadas na lista, *Message Control* consegue controlar as suas durações na lista do pares, permitindo assim ao *Message Control* combater *overhead* de mensagens na rede.

3.3.5 Transmissão e recepção de dados

O mecanismo de Transmissão e recepção de dados (representado por *Transmission/Reception of data* (figura 3.5)) tem como finalidade transmitir e receber os dados na rede. No presente trabalho utilizamos a transmissão e recepção de dados intra-grupo (*P2P Group*) e a transmissão e recepção de dados inter-grupos.

3.3.5.1 Transmissão e recepção de dados intra-grupo

A Transmissão e recepção de dados intra-grupo, baseia-se na transmissão e recepção de dados através do *framework Wi-Fi Direct*, onde utilizamos *sockets* TCP no transporte dos dados entre os nós na rede. Neste âmbito, foram utilizadas as seguintes técnicas:

- Todos os dispositivos num *P2P Group* (*P2P GO* e *P2P Client*) irão conter dois *ServerSocket* (*ServerSocket 1* e *ServerSocket 2*) abertos à escuta numa porta por conexões de *sockets* clientes. *ServerSocket 1* é utilizado na recepção de mensagens contendo *opUser* destinados a controlo de utilizadores e o *Serversocket 2* é utilizado na recepção de todas as mensagens não destinadas ao *User Control*.
- Cada *ServerSocket* irá conter uma porta distinta, consoante o papel desempenhado pelos seus dispositivo *Wi-Fi Direct* (*P2P GO* ou *P2P Client*).

  Nos dispositivos *P2P GO*, *ServerSocket 1* irá escutar na porta *x 1*, e *ServerSocket 2* na porta x3. E nos dispositivos *P2P Client*, *ServerSocket 1* irá escutar na porta *x 2*, e *ServerSocket 2* na porta *x 4*. Neste sentido, os *GOs* recebem mensagens sempre nas mesmas portas *x 1* (para mensagens de controlo de utilizador) e *x 3*, e transmitem também sempre para as mesmas portas *x 2* (para mensagens de controlo de utilizador) e *x 4*, já que todos os *P2P Clients* recebem nessas portas.

Em suma, optamos por realizar transmissões diretas de dados apenas entre GO e os seus *P2P Clients*. Portanto, as comunicações entre dois *P2P Clients* do mesmo grupo passam sempre pelo GO do grupo. As transmissões e recepção de dados entre o GO e o *P2P Client*, acontecem em paralelo, o GO transmite sempre para porta *x 2* e *x 4* (portas por onde *ServerSockets* dos *P2P Clients* estão à escuta de conexões TCP para receberem dados) e o *P2P Client* transmite sempre para porta *x 1* e *x 3* (portas por onde *ServerSockets* dos GOs estão à escuta de conexões TCP para receberem dados).

3.3.5.2 Transmissão e recepção de dados inter-grupos

Para transmissão e recepção de dados inter-grupos, utilizamos *BluetoothSocket* entre dois dispositivos *Bluetooth* que fazem parte de *P2P Group* diferentes. Nesta *framework* os dispositivos partilham o mesmo canal RFCOMM durante todo tempo que se mantêm conectados.

Capítulo 4

Implementação

O presente capítulo descreve a implementação da aplicação. Encontra-se dividido em duas secções, a primeira descreve a implementação do *middleware* de comunicação e os seus mecanismos, e a segunda secção descreve a aplicação para demonstração, onde é abordado a adaptação do *wondercom [30]* para utilizar o serviço disponibilizado pelo *middleware* de comunicação desenvolvido.

4.1 Middleware de comunicação

O *Middleware* de comunicação (*OpServiceConnection*), foi implementado para executar como um *Service Android*. Portanto estendemos essa classe, para que o *OpServiceConnection* execute operações durante um longo período de tempo em *background*, e disponibiliza-as ao *Client*.

Toda a implementação foi realizada através de *Android Studio*, que é o IDE (*Integrated Development Enviroment*) oficial para desenvolvimento de aplicações *Android*. Sendo a plataforma *Android* o alvo para a *App RO-WDB*, tivemos que definir a versão *Android* mínima que a suporta. Apesar do suporte à manipulação dos recursos *Wi-Fi Direct* ter sido introduzido na versão *Android 4.0*, utilizamos a versão *Android 4.1*, por ser a primeira a suportar o *Service Discovery* do *framework Wi-Fi Direct*, que é utilizado pelo mecanismo *Communication* no processo de descoberta de pares vizinhos com características desejadas.

Na implementação do *OpServiceConnection* utilizamos várias *Threads*, sobretudo nas transmissões e recepções de dados, e nos processos de estabelecer conexões *Wi-Fi Direct* e Bluetooth, por serem operações que podem levar um longo período a serem executados, bloqueando assim o *OpServiceConnection*, colocando-o à espera que a operação termine. Neste sentido, realizamos cada uma destas operações numa *thread* específica, para que estas possam realizar sua execução de forma concorrente, deixando o *OpServiceConnection* livre para executar outras operações.

4.1.1 Interfaces de comunicação com aplicações clientes

Para comunicação entre *OpServiceConnection* e as aplicações clientes, utilizamos o AIDL (*Android Interface Definition Language*) mecanismo este que permite ao programador definir uma interface de programação, onde o cliente e o *Service*, ambos acordam para comunicação entre si, a utilização de IPC (*Inter-Process Communication*), pois, o *OpServiceConnection* e o *Client* encontram-se em processos diferentes, e em *Android* normalmente um processo não consegue aceder a memória de um outro processo. Portanto, para comunicar terão que decompor os seus objetos em tipos de dados primitivos, os quais o sistema operativo consegue processar e transmitir para além do limite de um processo.

Assim sendo, definimos duas interfaces AIDL para comunicação entre o processo do *OpServiceConnection* e o do *Client*. A interface *Op-Input Interface* (figura 3.5 representada na arquitectara da aplicação) é usada na comunicação na direção *Client* para *OpServiceConnection*. E a interface *Client-Input Interface* é usada para a comunicação na direção contrária, ou seja, na direção *OpServiceConnection* para *Client*.

4.1.1.1 Op-Input Interface

A *Op-Input Interface* foi implementada para executar no *OpServiceConnection*. Contém diversos métodos que permitem ao *Client* solicitar ao *OpServiceConnection* a execução de uma determinada tarefa, podendo passar dados como parâmetro. Um exemplo a citar é o método *sendMessage(OpPackage)* que permite ao *Client* enviar um *OpPackage* como parâmetro para o *OpServiceConnection*, solicitando-o que a envie a outros utilizadores na rede Oportunista.

Sendo esta interface uma interface AIDL, para que o *Client* esteja habilitado a comunicar com o *OpServiceConnection*, através desta interface, terá que:

1. Ter uma cópia do ficheiro ".aidl" desta interface na sua diretoria.
2. Obter uma instância do *Op-Input Interface*, de forma a ser possível chamar os seus métodos, solicitando assim, ao *OpServiceConnection* que as executem, sendo que os dados podem ser enviados ao *OpServiceConnection*, como parâmetros dos métodos chamados.

O *Client*, ao ter do seu lado, uma cópia do ficheiro .aidl e uma instância do *Op-Input Interface*, ficará automaticamente habilitado a inicializar comunicação com o *OpServiceConnection*.

4.1.1.2 Client-Input Interface

Na *Client-Input Interface* são implementados vários métodos, mais concretamente no *ConnectionToService*, que permitem ao *Client* receber as comunicações provenientes do *OpServiceConnection*. Um exemplo desta comunicação são as mensagens recebidas num dispositivo, no *OpServiceConnection*, que através do mecanismo *Message Control* irá verificar se a mensagem é destinada

a esse dispositivo. Caso o for, irá encaminhar a mensagem para o *Client* através do método *onReceiveMessage(OpPackage)* da *Client-Input Interface*, onde *OpPackage* (mensagem) é enviada como parâmetro.

Já que o *Client-Input Interface* é uma interface AIDL, para que o *OpServiceConnection* esteja habilitado a inicializar a comunicação com o *Client*, através desta interface, terá que:

1. Ter uma cópia do ficheiro .aidl desta interface na sua diretoria.
2. Obter uma instância do *Client-Input Interface*, de forma a ser possível chamar os seus métodos, solicitando assim ao *Client* que as executem, sendo que os dados são enviados como parâmetros dos métodos chamados. A instância desta interface é representada por *Op-Output Interface* na arquitetura da aplicação.

O *OpServiceConnection*, tendo uma cópia do ficheiro .aidl e uma instância do *Client-Input Interface* (*Op-Output Interface*), já estará habilitado a inicializar comunicação com o *Client*.

4.1.2 ConnectionToService

O *ConnectionToService* foi implementado como uma classe java, onde se encontram todos os mecanismos necessários para permitir comunicações (bidirecionais) seguras entre *OpServiceConnection* e o *Client*.

A partir desta classe, o *Client* consegue realizar *bindService()* ao *OpServiceConnection*, tendo *BIND_AUTO_CREATE* como modo de funcionamento para efetuar a ligação, o que significa que *OpServiceConnection* será criado automaticamente, mantendo-se a correr durante todo o tempo que a ligação se mantiver. Através do *bindService()* o *Client* obtém uma instância da *Op-Input Interface* que lhe permitirá inicializar comunicação com *OpServiceConnection*. De forma a ser segura as chamadas aos métodos da *Op-Input Interface* através da sua instância, criamos uma API com métodos que irão auxiliar nas chamadas de cada método presente na instância da *Op-Input Interface*. Esta API encontra-se representada por *Api Access Middleware* na arquitetura da aplicação.

Além disso, nesta classe, é implementado a *Client-Input Interface* que é a porta de entrada das comunicações provenientes do *OpServiceConnection* no *Client*. Para que o *OpServiceConnection* possa enviar dados a porta de entrada do *Client*, terá que obter uma instância da *Client-Input Interface* (*Op-Output Interface*), a qual implementamos para que seja obtido logo na inicialização do *OpServiceConnection*, através do método *registerListener()* onde a instância da *Client-Input Interface* (*Op-Output Interface*) é enviado como o seu parâmetro.

O *registerListener()* é chamado logo após a inicialização do *OpServiceConnection*. Ele é um dos métodos da *Op-Input Interface*, o qual é passado uma instância desta interface como parâmetro, de modo a ser entregue ao *OpServiceConnection*.

O *registerListener()*, é um dos métodos da *Op-Input Interface*, tem como função registar o *Client* que liga ao *OpServiceConnection*. Sendo assim, são guardados no *OpServiceConnection* os dados (a *string* com o nome do pacote do *Client* (isto de forma a ser um identificador único para cada *Client*) e a instância da *Client-Input Interface* (*Op-Output Interface*)) passados como parâmetro do método *registerListener()*.

4.1.3 Mensagens

Para a implementação de uma mensagem (*OpPackage*) na rede Oportunista e do *OpUser* que é transmitido nele, criamos *OpPackage* e *OpUser* como um objecto *Serializable* e *Parcelable*, em simultâneo. Ou seja, tanto na classe *OpPackage* como na *OpUser*, implementamos ao mesmo tempo as interfaces *Parcelable* e *Serializable*, de forma a ser possível o *marshaling* e *unmarshaling* dos objetos *OpUser* e *OpPackage*, tanto nas comunicações entre os componentes da aplicação (dentro do dispositivo), como nas comunicações entre dispositivos.

Utilizamos *Serializable* para realizar *marshaling* e *unmarshaling* dos objetos, nas comunicações de dispositivo para dispositivo, uma vez que os objetos *Serializable* são suportados pelo *OutputStream* e *InputStream* que são os mecanismos responsáveis pelo envio e recepção de dados nas comunicações através de *Sockets* e de *BluetoothSockets*. Os objetos podem ser serializados através do *ObjectOutputStream* e des-serializados através do *ObjectInputStream*.

Serializable é uma interface padrão da linguagem Java. Funciona como um objecto que é convertido em *stream* de *bytes*, e depois reconvertido para objecto. Para implementar um objeto como *Serializable* é preciso apenas adicionar *implements Serializable* na declaração da classe Java, não havendo necessidade de implementar qualquer método. Em *Serializable* a operação *marshaling* é realizado no *Java Virtual Machine* (JVM) utilizando Java *reflection* API que dá o melhor para identificar os dados membros dos objetos nas aplicações *Android*, o que resulta em vários objetos temporários. Deste modo, o processo de *Serializable* é lento em comparação com *Parcelable*.

Já *Parcelable* é uma interface específica da plataforma *Android*. Foi utilizada para realizar *marshaling* e *unmarshaling* dos objetos nas comunicações dentro dos dispositivos, ou seja, entre os componentes das aplicações *Android*. Quando um objecto é enviado como sendo *Parcelable*, este é convertido em tipos primitivos para depois ser reconstruído. Implementar um objecto *Parcelable* é mais trabalhoso quando comparado com *Serializeble*, visto que é preciso implementar alguns métodos e escrever o código específico de cada objecto que se pretende enviar como sendo *Parcelable*. Com esta abordagem, *Parcelable* utiliza menos objetos temporários no processo de *marshaling*, já que o código para transformar cada dado membro de um objecto é escrito pelo programador [33].

4.1.4 Mecanismo de comunicação

Na implementação do mecanismo de comunicação (*Communication*), utilizamos as *APIs Wi-Fi Direct* e *Bluetooth* disponibilizados pelo *SDK Android*. Os apêndices C (*APIs Wi-Fi Direct*) e B (*APIs Bluetooth*), apresentam uma breve descrição dessas *APIs*, do modo como funcionam, e uma pequena descrição dos seus métodos, classes e *Listeners* (Interface para invocar *callback*, quando acontece uma determinada ação). É de salientar que na maior parte dos métodos das *APIs* utilizadas, criamos uma *Thread* distinta para cada um, uma vez que são métodos bloqueantes.

Para além dessas *APIs* utilizadas na implementação do mecanismo de comunicação, criamos *broadcast receiver* para cada um dos *frameworks (Wi-Fi Direct e Bluetooth)*, onde registamos os seus respectivos *Intents* (como exemplo: *WIFI_P2P_STATE_CHANGED_ACTION* para *Wi-Fi Direct* e *ACTION_CONNECTION_STATE_CHANGED* para *Bluetooth*), de forma a permitir ao *Communication* receber notificações de eventos do seu interesse, como por exemplo: saber se o *Wi-Fi Direct*, *Bluetooth* dos dispositivos encontram-se ligados/desligados, saber se um dispositivo perdeu uma conexão *Wi-Fi Direct*/*Bluetooth*, etc.

4.1.4.1 WifiP2pManager

A partir das *APIs Wi-Fi Direct* (ou *APIs Wi-Fi Peer-to-Peer*) desenvolvemos as classes e métodos necessários para que o mecanismo de comunicação possa usufruir dos recursos fornecidos pelo *framework Wi-Fi Direct*. Quando o *OpServiceConnection* é inicializado, é obtido uma instância do objecto *WifiP2pManager* que será utilizado para registar a aplicação ao *framework Wi-Fi Direct* através da chamada do método *initialize()*, que irá retornar um *WifiP2pManager.Channel*, que será utilizado para ligar a aplicação ao *framework Wi-Fi Direct*. Tendo esses dois objetos *WifiP2pManager* e *WifiP2pManager.Channel*, o mecanismo de comunicação já estará apto a chamar os métodos da classe *WifiP2pManager* para poder usufruir das funcionalidades fornecidas pelo *framework Wi-Fi Direct*.

Na conexão entre pares utilizamos o método *connect()*. Portanto, os dispositivos terão de descobrir os pares a que se desejam conectar, antes de chamar o método *connect()*, e na descoberta de pares utilizamos o mecanismo de descoberta de serviço descrito na subsecção C.1.2.1 do apêndice C.

4.1.4.2 BluetoothAdapter

Através das *APIs Bluetooth* desenvolvemos as classes e métodos necessários para que o mecanismo de comunicação possa usufruir dos recursos fornecidos pelo *framework Bluetooth*. O objeto *BluetoothAdapter* é o meio pelo qual uma aplicação consegue interagir com o *framework Bluetooth*; representa o Adaptador *Bluetooth* Local. É obtido também na aplicação assim que o *OpServiceConnection* é inicializado, através da chamada ao método *getDefaultAdapter()* da classe *BluetoothAdapter*. Havendo uma instância do objecto *BluetoothAdapter*, já será possível

ao mecanismo de comunicação usufruir das funcionalidades fornecidas pelo *framework Bluetooth*, como por exemplo: permitir ser descoberto por outros dispositivos, realizar a descoberta de pares, conectar-se com pares encontrados, etc.

Ao ser obtida uma instância do objecto *BluetoothAdapter* logo na inicialização do *OpService-Connection*, o *OpServiceConnection* fica habilitado a usufruir das funcionalidades do *framework Bluetooth*. Optamos por realizar na inicialização do *OpServiceConnection*, apenas a operação que permite ao dispositivo ser descoberto por outros dispositivos, por ser uma operação que exige a intervenção do utilizador, para aceitar ou não que o seu dispositivo seja descoberto por dispositivos vizinhos. Os processos necessários para que um dispositivo *Bluetooth* possa ser descoberto sempre (durante todo o tempo) por dispositivos *Bluetooth* vizinhos, encontram-se descritos na secção B.2 do apêndice B.

A técnica de implementação que utilizamos para realizar conexões entre dispositivos *Bluetooth* de modo automatizado, é preparar automaticamente cada dispositivo como um servidor, de forma que cada um tenha um *socket* aberto, à escuta de conexões de *sockets* clientes. Desta forma, todos os dispositivos estarão aptos para aceitar uma conexão de outros dispositivo (neste caso o dispositivo irá desempenhar o papel de *Master* na *Piconet*), assim como, iniciar uma conexão com um dispositivo remoto descoberto (neste caso o dispositivo terá o papel de *Slave* na *Piconet*).

4.2 Aplicação para demonstração

Como aplicação para demonstração utilizamos *wondercom [30]*, que é uma aplicação de chat que permite aos utilizadores trocar mensagens de texto, imagem, áudio e vídeo, dentro de um *P2P Group* que já se encontra formado. Na implementação aproveitamos toda a interface gráfica de wondercom, e os seus mecanismos para gravação de áudio e vídeo em ficheiros, tirar fotografias, bem como os mecanismos para apresentação de listas de objetos (mensagens) numa *ListView Android*. Além disso, acrescentamos uma classe *ROService* a qual estendemos a classe *Service* do *Android* de forma a poder executar em *background*, a interagir com o *middleware* de comunicação (OpServiceConnection), mas a executar no mesmo processo que os restantes componentes do *wondercom*.

Para que seja possível a interação com o *OpServiceConnection* a partir do *ROService*, colocamos o ficheiro ".java" da classe *ConnectionToService* na mesma diretoria onde encontram os ficheiros de execução da *App wondercom*, e criamos uma instância do objecto *ConnectionToService* em *ROService*. Tendo uma instância do objeto *ConnectionToService* em *ROService*, já será possível interagir com o *OpServiceConnection*, uma vez que, o *ConnectionToService* tem nele implementado todos os mecanismos, métodos e interfaces necessários para que haja uma comunicação bidirecional com o *OpServiceConnection*. Neste sentido os utilizadores poderão receber notificações e mensagens (provenientes de outros dispositivos que são recebidas no OpServiceConnection e reencaminhadas ao *Client*) do *OpServiceConnection*, mesmo estando o

utilizador com a tela (*Activity*) da aplicação fechada.

No *ROService* ao ser recebido uma notificação ou mensagens do OpServiceConnection, são reencaminhadas as *Activitys* do *wondercom*, mas sem atualizar a interface gráfica do utilizador (Android UI toolkit). Para proceder sua atualização criamos uma classe *showMessageRunnable* que implementa a interface *Runnable* que irá conter operações para atualizar a interface gráfica do utilizador. Isto possibilita a execução das operações na *UI Thread* garantindo que *Android UI toolkit* não será acedido de fora da *UI Thread* (aplicação *thread-safe*). Para a execução da *showMessageRunnable* na*UI Thread*, é chamado o método *runOnUiThread()* da *Activity* e passado a interface *showMessageRunnable* como parâmetro.

Ao nível gráfico da aplicação *wondercom*, fizemos uma pequena alteração na tela do chat, o qual acrescentamos:

1. Um *Switch* (com o nome de RO), para inicializar e desligar o *OpServiceConnection*. *Switch* é um *widget* do *Android OS* que funciona como um interruptor entre dois estados, possibilitando a escolha entre duas opções.

2. Um botão (como o nome Net Data), para solicitar ao *OpServiceConnection*, as informações do dispositivo na rede.

3. Um *TextView*, que é um *widget* do *Android OS* para apresentar textos ao utilizador. Este, dependendo do estado de execução do *OpServiceConnection*, irá alternar entre os seguintes textos: *inactive*, *scan WD*, *scan B* e *expanded*. O texto *inactive* será apresentado no *TextView* sempre que o *OpServiceConnection* estiver desligado. O texto *scan WD* é apresentado durante os momentos em que o dispositivo se encontra nos estados *Discovery Wi-Fi Direct Network* e *Try Connection*, e o texto *scan B* durante os estados *Discovery More Network To Make Bluetooth Tunel* e *Try Bluetooth Connection To Make Tunel To Expand Network*. E por ultimo, o texto *expanded* é apresentado no *TextView* quando o dispositivo estiver no estado *Bluetooth Connected isTunel Network Expanded*.

4. Ao lado do *TextView* da alínea anterior, é igualmente adicionado um outro *TextView* contendo texto com apenas uma letra, que pode variar entre a letra C e a D. A letra C é apresentado no *TextView* sempre que o dispositivo se encontra conectado numa rede do *OpServiceConnection*, e a letra D é apresentado caso contrário.

5. Logo abaixo dos *widgets* abordados nos itens anteriores, adicionamos um *TextView* de cor cinzento claro, que irá apresentar a *string* com as informações do dispositivo adquerido apartir das solicitações realizadas pelo botão Net Data, abordado no item 2. Esta solicitação retorna uma *string* com o texto "Dados de rede"quando o *OpServiceConnection* estiver desligado, e caso contrário, retorna uma string com a informação do estado do dispositivo. Esta string contém seis campos "chave:valor"separados por ";", sendo que o valor apenas é apresentado caso existe. Por exemplo o texto "id: ; Group: ; isGO: ; userTunnel: ; groupTunnel: ; p2pState: ". No texto, cada campo irá conter os seguintes valores: I) *id* - conterá o id do *OpUser* do próprio dispositivo; II) *Group* - irá conter o id do seu GO;

III) *isGO* - conterá *true* caso o dispositivo for GO, e *false* caso contrário; IV) *userTunnel* - conterá o *userTunnel* do seu *OpUser*; V) *groupTunnel* - conterá o *groupTunnel* do seu *OpUser*; VI) *p2pState* - conterá o *p2pState* do dispositivo.

Para além das alterações que realizamos na tela do chat, também, criamos uma tela (*Activity*) para que os utilizadores possam alterar os seus dados. Esta tela encontra-se ilustrada na figura 4.6.

As figuras abaixo contêm *screenShots* do RO-WDB em diferentes estados de execução, incluindo exemplos de envios e recepções de dados entre dispositivos conectados na rede do *OpServiceConnection*. É de salientar que na tela do chat, as mensagens recebidas/enviadas são apresentadas no *ListView*, dentro de um retângulo. As mensagens enviadas pelo dispositivo são apresentadas com fundo branco, e as mensagens recebidas no dispositivo são apresentadas com fundo azul claro.

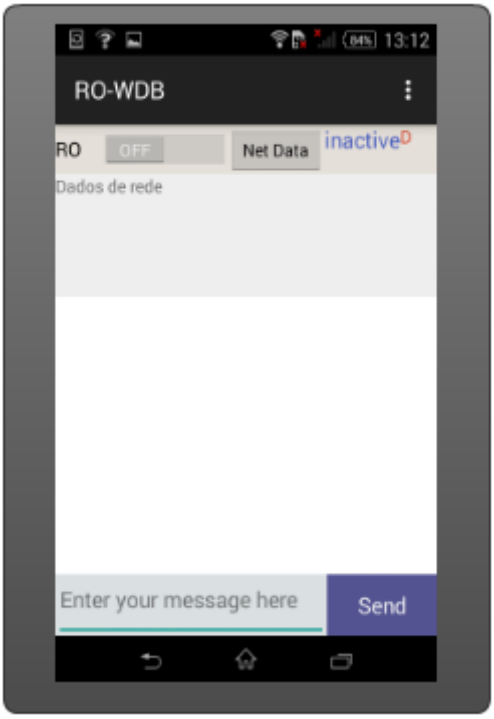


Figura 4.1: Tela do chat com o Serviço de Comunicação desligado

Figura 4.2: Serviço middleware Comunicação a realizar procura de redes próximos

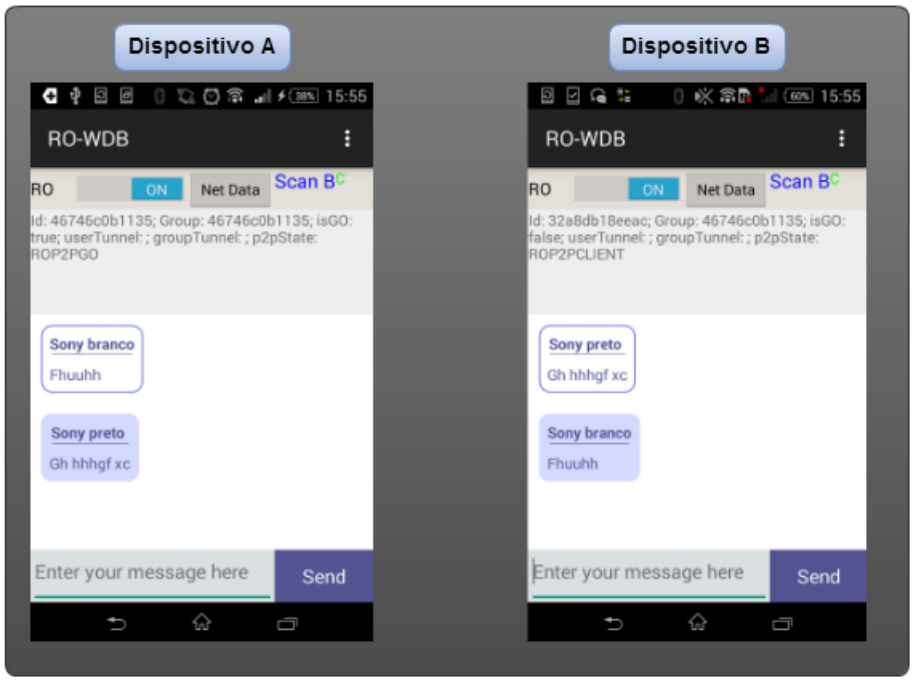


Figura 4.3: Transferência de dados (texto simples) entre dispositivos do mesmo grupo

Figura 4.4: Transferência de dados (vídeo) entre dispositivos do mesmo grupo

Figura 4.5: Transferência de dados (ficheiro de qualquer tipo) entre dispositivos do mesmo grupo

Figura 4.6: Tela para editar os dados do utilizador

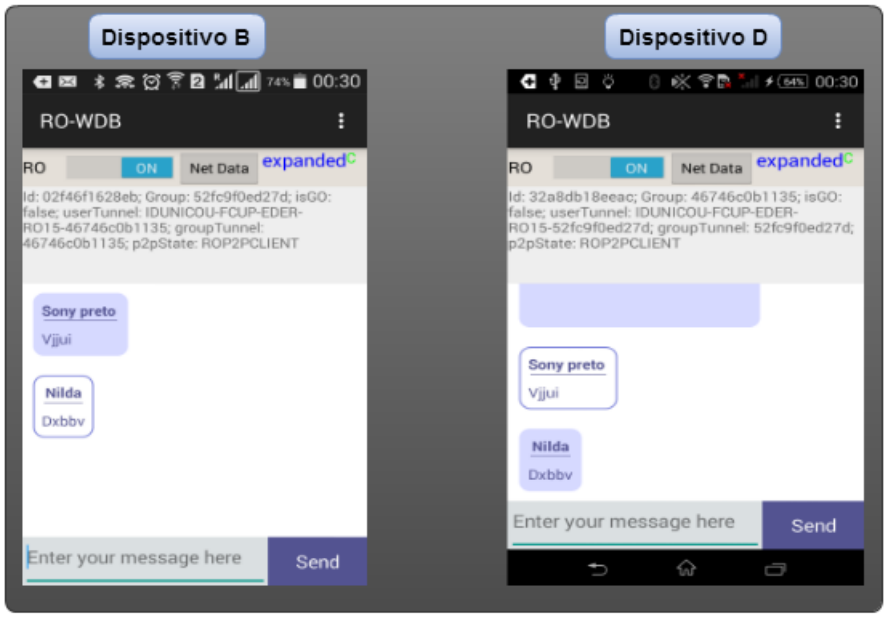


Figura 4.7: Transferência de dados entre dispositivos de grupos diferentes

Capítulo 5

Conclusões

Nesta dissertação foi implementado um *middleware* de comunicação que permite aos dispositivos *Android* que suportam *framework Wi-Fi Direct* e *Bluetooth*, colaborarem entre si na construção de uma rede oportunista, bem como comunicação entre os nós da rede. A rede construída pode conter um ou mais grupos *Wi-Fi Direct*, onde a interligação física entre esses grupos dá-se através de conexões *Bluetooth* entre um dos pares de cada grupo. Essa abordagem, permitiu estender o alcance de comunicação da rede, que não seria alcançável pelos protocolos de comunicação D2D (*Wi-Fi Direct* e *Bluetooth*) disponíveis nos dispositivos *Android "non-rooted"* , uma vez que, estes foram adaptados apenas para um único grupo de comunicação D2D.

De uma forma mais específica, procurou-se solucionar as limitações das topologias de redes formadas por dispositivos *Wi-Fi Direct* e *Bluetooth*, nomeadamente na abrangência a uma área de cobertura superior às suportadas por um grupo formado por esses dispositivos, como é o caso de *P2P Group* para dispositivos *Wi-Fi Direct*, e *Piconet* para *Bluetooth*. Foi projetada a construção de uma topologia lógica por cima de redes físicas *P2P Group* e *Piconet*, a qual optamos por utilizar *P2P Group* como uma sub-rede (ou região, ou grupo) da rede no seu todo, e utilizamos uma *Piconet* simples constituída apenas por uma conexão entre dois dispositivo *Bluetooth*, para ligar fisicamente duas sub-redes. Nesta linha, uma *Piconet* será formada entre dois dispositivos que pertencem a sub-redes diferentes e desta forma irão servir de *gateway* das suas sub-redes para outras. Todo o encaminhamento de mensagens na rede é realizado na camada de aplicação, a qual empregamos a técnica *store-carry-forward* de mensagens, que consiste em fazer com que no envio de uma mensagem da origem ao destino, cada nó intermediário irá guardar a mensagem que recebeu, guardando-a até encontrar um possível nó para a reencaminhar, conseguindo deste modo lidar com a intermitência de conexão existente nesse tipo de rede.

O *middleware* implementado nesta dissertação dá às aplicações clientes (como chat, jogos *multi players*, redes sociais) a possibilidade de fornecerem aos seus utilizadores os seus serviços funcionando numa rede oportunista, sendo que toda a parte de conexão, manutenção e comunicação na rede oportunista é realizado pelo *middleware*. Às aplicações clientes conseguem interagir com o middleware, usufruir dos seus serviços, através da utilização de uma camada adicional de nome

ConnectionToService, que é um mecanismo/biblioteca de acesso ao *middleware*. E o *middleware* realiza a construção automática da rede oportunista. Isto é conseguido pelo aproveitamento da capacidade de descoberta de pares vizinhos, existente nas tecnologias *Wi-Fi Direct* e *Bluetooth*, para a descoberta de pares e redes próximos/desejados, e consequentemente estabelecer conexões sem que haja a intervenção de utilizadores. O *middleware* permite igualmente a comunicação entre todos os nós (dispositivos) da rede, controla as entradas/saídas e atualização de dados de todos os utilizadores da rede. E ainda controla a duração de uma mensagem na rede, combatendo assim o *overhead* de mensagens na rede.

A construção automatizada da topologia de rede é uma das mais valias do *middleware* implementado, já que os utilizadores não terão de realizar tarefas como procurar por uma rede disponível, configurar os dispositivos para que possam realizar conexões a uma rede, entre outros. Também tem uma desvantagem para os utilizadores, uma vez que os dispositivos, sobretudo no processo de expansão da rede, efetuam repetidamente descoberta de pares (descoberta de outro grupo para realizar a expansão da rede), enquanto não for estabelecido uma conexão, consumindo assim muita bateria dos dispositivo. Para além da construção automatizada da topologia de rede, a técnica de encaminhamento de mensagens utilizada também pode ser uma inconveniência para os utilizadores, principalmente quando os seus dispositivos funcionam como GO, pois, na técnica de encaminhamento de mensagens implementada, as comunicações dentro dos grupos passam sempre pelos seus respectivos *GOs*, o que pode sobrecarregar esses dispositivos (sobretudo quando esses *GOs* têm vários dispositivos como seus *Clients*).

5.1 Trabalho Futuro

O nosso *middleware* de comunicação para Redes Oportunistas abre várias possibilidades de estudos e pesquisas em diferentes direções, que podem trazer melhorias e benefícios ao *middleware* implementado. Abaixo segue algumas sugestões de trabalhos futuros que podem ser realizados com base nesta dissertação:

- Realizar um estudo aprofundado para determinar a escalabilidade do sistema: observar o número de nós que a rede e as sub-redes (grupos) possam eventualmente suportar; o tempo de atraso no encaminhamento das mensagens da origem ao destino; até que ponto se pode expandir a rede e manter ao mesmo tempo uma qualidade de serviço aceitável; entre outros.
- Implementação de mecanismo que utiliza a interface *Wi-Fi* dos dispositivos *Android* de forma a realizar conexão a WLAN (*Wireless Local Area Network*), possibilitando assim a expansão da rede à Internet.
- Estudar a possibilidade de se introduzir comunicações em tempo real. Um possível estudo nesse sentido seria verificar se existe possibilidade de ser integrado (e a forma de o fazer) comunicação em tempo real no *middleware*, utilizando *P2PSIP protocol* que é um protocolo

que habilita a comunicação em tempo real utilizando SIP (*Session Initiation Protocol*) de um modo *peer-to-peer*.

- Modificar o mecanismo de controlo de mensagens para realizar encaminhamento de mensagens de forma mais eficiente.
- Realizar um estudo aprofundado relativamente à segurança da rede construída.
- Realizar estudos para possibilitar o anonimato dos utilizadores na rede, uma vez que, as mensagens encaminhadas nesse tipo de redes podem passar por vários nós intermediários até ao destino. Uma possível técnica a ser estudado é *Onion routing*, que é uma técnica para comunicação anonima em rede de computadores, a qual impede que os nós intermediários descubram a identidade daqueles que andam a comunicar através deles.

Bibliografia

[1] Android Developer. Bluetooth. Online, February 2015. http://developer.android.com/guide/topics/connectivity/bluetooth.html.

[2] Android Developer. Wi-fi peer-to-peer. Online, February 2015. http://developer.android.com/guide/topics/connectivity/wifip2p.html.

[3] Marco Conti and Silvia Giordano. Multihop ad hoc networking: The theory. *Communications Magazine, IEEE*, 45(4):78–86, 2007.

[4] Annapurna P Patil, Narmada Sambaturu, and Krittaya Chunhaviriyakul. Convergence time evaluation of algorithms in MANETs. *arXiv preprint arXiv:0910.1475*, 2009.

[5] Houda Labiod, Hossam Afifi, and Costantino De Santis. *WI-FITM, BluetoothTM, ZigbeeTM and WiMAXTM*. Springer Science & Business Media, 2007.

[6] Wi-Fi Alliance. Wi-fi certified wi-fi direct. *White paper*, 2010.

[7] Daniel Camps-Mur, Andres Garcia-Saavedra, and Pablo Serrano. Device-to-device communications with Wi-Fi Direct: overview and experimentation. *Wireless Communications, IEEE*, 20(3):96–104, 2013.

[8] Marco Conti, Franca Delmastro, Giovanni Minutiello, and Roberta Paris. Experimenting opportunistic networks with WiFi Direct. In *Wireless Days (WD), 2013 IFIP*, pages 1–6. IEEE, 2013.

[9] W Keith Edwards. Discovery systems in ubiquitous computing. *Pervasive Computing, IEEE*, 5(2):70–77, 2006.

[10] Raffaele Bruno, Marco Conti, and Enrico Gregori. Bluetooth: Architecture, protocols and scheduling algorithms. *Cluster Computing*, 5(2):117–131, 2002.

[11] SIG Bluetooth. Bluetooth core specification version 4.0. *Specification of the Bluetooth System*, 2010.

[12] Jaap Haartsen. Bluetooth-the universal radio interface for ad hoc, wireless connectivity. *Ericsson review*, 3(1):110–117, 1998.

[13] Bluetooth tutorial. Online, 2015. http://www.tutorial-reports.com/wireless/bluetooth/protocolstack.php.

[14] Christian Bettstetter. On the minimum node degree and connectivity of a wireless multihop network. In *Proceedings of the 3rd ACM international symposium on Mobile ad hoc networking & computing*, pages 80–91. ACM, 2002.

[15] Yan Zhang, Jijun Luo, and Honglin Hu. Comparison between wireless ad hoc and mesh networks. In *Wireless mesh networking: architectures, protocols and standards*, pages 22–24. CRC Press, 2006.

[16] Josh Broch, David A Maltz, David B Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th annual ACM/IEEE international conference on Mobile computing and networking*, pages 85–97. ACM, 1998.

[17] David Rodriguez Guilherme Iecker, Fabio Antunes. Redes sensores sem fio. Online, August 2015. http://www.gta.ufrj.br/grad/12_1/rssf/def.html.

[18] ebase. Redes mesh. Online, August 2015. http://ebase.webnode.com.br/news/ebase/.

[19] Myung J Lee, Jianliang Zheng, Young-Bae Ko, and Deepesh Man Shrestha. Emerging standards for wireless mesh technology. *Wireless Communications, IEEE*, 13(2):56–63, 2006.

[20] S. Tarkoma. *Overlay Networks: Toward Information Networking.* CRC Press, 2010.

[21] Claudio Casetti, Carla-Fabiana Chiasserini, Luciano Curto Pelle, Carolina Del Valle, Yufeng Duan, and Paolo Giaccone. Content-centric routing in wi-fi direct multi-group networks. *arXiv preprint arXiv:1412.0880*, 2014.

[22] Anshul Verma, Dr Srivastava, et al. Integrated routing protocol for opportunistic networks. *arXiv preprint arXiv:1204.1658*, 2012.

[23] Luciana Pelusi, Andrea Passarella, and Marco Conti. Opportunistic networking: data forwarding in disconnected mobile ad hoc networks. *Communications Magazine, IEEE*, 44(11):134–141, 2006.

[24] Chung-Ming Huang, Kun-chan Lan, and Chang-Zhou Tsai. A survey of opportunistic networks. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 1672–1677. IEEE, 2008.

[25] Isai Michel Lombera, Louise E Moser, PM Melliar-Smith, and Yung-Ting Chuang. Peer management for iTrust over Wi-Fi Direct. In *Wireless Personal Multimedia Communications (WPMC), 2013 16th International Symposium on*, pages 1–5. IEEE, 2013.

[26] Tuan Nguyen Duong, Ngoc-Thanh Dinh, and YoungHan Kim. Content sharing using P2PSIP protocol in Wi-Fi direct networks. In *Communications and Electronics (ICCE), 2012 Fourth International Conference on*, pages 114–118. IEEE, 2012.

[27] Yufeng Duan, Carlo Borgiattino, Claudio Casetti, Carla Fabiana Chiasserini, Paolo Giaccone, Marco Ricca, Fabio Malabocchia, and Maura Turolla. Wi-fi direct multi-group data

dissemination for public safety. In *WTC 2014; World Telecommunications Congress 2014; Proceedings of*, pages 1–6. VDE, 2014.

[28] Briar. Briar how it works. Online, December 2014. https://briarproject.org/how-it-works.html.

[29] Open Garden. Open garden our family of apps. Online, December 2014. https://opengarden.com/.

[30] Rubeus. Wondercom. Online, April 2014. https://github.com/rubeus90/WonderCom.

[31] Jin-Shyan Lee, Yu-Wei Su, and Chung-Chou Shen. A comparative study of wireless protocols: Bluetooth, uwb, zigbee, and wi-fi. In *Industrial Electronics Society, 2007. IECON 2007. 33rd Annual Conference of the IEEE*, pages 46–51. IEEE, 2007.

[32] Ling Pei, Ruizhi Chen, Jingbin Liu, Heidi Kuusniemi, Tomi Tenhunen, and Yuwei Chen. Using inquiry-based bluetooth rssi probability distributions for indoor positioning. *Journal of Global Positioning Systems*, 9(2):122–130, 2010.

[33] Kristina Ferm. Developing android application protoype for internal social networks for companies. *http://www.diva-portal.org*, 2014.

[34] Android Developer. Application fundamentals. Online, February 2015. http://developer.android.com/guide/components/fundamentals.html.

[35] Android Developer. Activity. Online, February 2015. http://developer.android.com/guide/components/activities.html.

[36] Android Developer. Services. Online, February 2015. http://developer.android.com/guide/components/services.html.

[37] Android Developer. Content providers. Online, February 2015. http://developer.android.com/guide/topics/providers/content-providers.html.

[38] Android Developer. Processes and threads. Online, February 2015. http://developer.android.com/guide/components/processes-and-threads.html.

[39] Android Developer. Android interface definition language (aidl). Online, February 2015. http://developer.android.com/guide/components/aidl.html.

[40] S. Cheshire and M. Krochmal. Dns-based service discovery. Online, February 2013. http://www.ietf.org/rfc/rfc6763.txt.

Apêndice A

Android

Android é um sistema operativo móvel de código aberto, baseado no *Kernel Linux*. Pode-se dizer que é um sistema *Linux* de múltiplos utilizadores, em que cada aplicação representa um utilizador no sistema. Em *Android* cada processo têm a sua própria máquina virtual, portanto os códigos de aplicações diferentes correm separados uns dos outros. Por defeito um sistema *Android*:

- Atribui a cada aplicação um único *Linux User ID*, sendo que esse *ID* não é conhecido pela aplicação e é utilizado apenas pelo sistema Operativo.
- Cada aplicação corre no seu próprio processo *Linux*, assim como acontece nos demais sistemas operativos *Linux*. No *Android* o sistema inicia os processos quando qualquer um dos componentes da aplicação precisarem ser executados, e termina os processos quando não forem mais necessários ou quando o sistema precisa de recuperar memória para outras aplicações.

O *Android OS* é implementado com base no princípio de menor privilégio, em que por defeito cada aplicação tem acesso apenas aos componentes requeridos para realizar a tarefa. Desta forma o sistema torna-se menos vulnerável, onde uma aplicação não consegue ter acesso a partes de sistema em que não lhe foi dado permissão para o fazer. Contudo existem formas que permitem a uma aplicação partilhar dados com outras aplicações, e ter acesso a Serviços do sistema[34].

A.1 Componentes das Aplicações

No sistema operativo *Android* as aplicações são construídas essencialmente através dos componentes, onde cada componente pode representar um possível ponto de entrada do sistema. Existem quatro tipos de componentes: [34]:

- ***Activity*** – tem por objetivo a interação com o utilizador, é aqui que é implementado a

interface gráfica do utilizador. Em cada *Activity* é fornecida uma tela na qual se pode desenhar a interface gráfica do utilizador[35].

- ***Service*** - permite a execução de tarefas em *background*, não fornece interface gráfica. É utilizado quando se pretende realizar operações que irão correr durante um longo período. O *Service* pode ser inicializado por outros componentes, entretanto, este continuará a correr mesmo que o componente que o inicializou seja destruído [36].

- ***Broadcast Receiver*** - Este componente responde a todos os anúncios de *broadcast* do sistema, dado que muitos dos *broadcast* tem como origem o próprio sistema, tendo como exemplo, *brodcast* a anunciar que a bateria está fraca, ou que uma imagem foi capturada. Aplicações podem também inicializar *broadcast* com o intuito de notificar outras aplicações que foi realizado download de algum dado no dispositivo e/ou esses dados já se encontram disponíveis para serem utilizados, etc. Com isso o *broadcast receiver* irá interceptar a comunicação e inicializar a ação apropriada[34].

- ***Content Provider*** – visa administrar a partilha dos conjuntos de dados da aplicação. Permite guardar os dados em ficheiros no sistema, em base de dados *SQLITE*, na *web* ou em qualquer outro local persistente de armazenamento no qual a aplicação possa aceder. Através deste componente outras aplicações podem consultar dados desta aplicação ou até mesmo modificá-las caso tenha permissão [37].

A.2 Processos e Threads

O *Android* permite ao sistema criar um processo Linux e um *thread* único de execução para cada aplicação que é executada. Quando um componente de uma aplicação é inicializado e não existe qualquer outro componente dessa aplicação a correr é criado um processo Linux e um *thread* único de execução, caso esse processo já exista, é inicializado o componente dentro desse processo e utilizado o mesmo *thread* de execução. Pois, por defeito em *Android* todos os componentes de uma aplicação irão correr no mesmo processo e no mesmo *thread* denominado *"main thread"*. Todavia, é possível determinar, através do ficheiro *manifest*, que diferentes componentes de uma aplicação correm em processos separados. Também pode-se criar *thread* adicionais para qualquer processo[38].

A.3 Threads

Conforme mencionado no ponto anterior, quando se inicia uma aplicação *Android*, o sistema cria um *thread* de execução (*"main thread"*, também conhecido como *UI thread*) para todos componentes da aplicação. Não é criada uma *thread* separada para cada instância dos componentes. Todos os componentes que correm no mesmo processo Serão instanciados no *UI thread*, que é a origem da expedição de todas as chamadas do sistema para cada componente. À vista disso,

o *UI Thread* é de crucial importância, visto que, é nesta camada que correm todos os métodos responsáveis por responder *callbacks* do sistema (como por exemplo: *nkeyDown()* para reportar ação do utilizador, ou chamar o método *callback* do ciclo de vida). Esta *Thread* permite inclusive despachar os eventos para os apropriados *widgets* da interface do utilizador. Permite a aplicação interagir com os componentes de *Android UI toolkit* (componentes do pacote *android.widget* e *android.view*)[38].

A.4 Thread-safe

De modo a visar pela boa performance e uma correta implementação de uma aplicação *Android*, deve-se garantir que a aplicação não irá realizar na *UI Thread*, operações de longa duração (como exemplo download, consultas a base de dados), de forma a evitar o bloqueio do mesmo. Contudo é vital para a capacidade de resposta da interface de utilizadores (UI) da aplicação, que não se bloqueia a *UI thread*. Posto isto, quando for preciso realizar operações que não são instantâneos, estes devem ser realizadas em *threads* separados ("*background*" ou "*worker threads*"). Pois, para construir/produzir uma aplicação *thread-safe* no *Android*, o programador para além de ter cuidado em não bloquear a *UI Thread*, também terá que garantir, que na aplicação, a *Android UI toolkit* não será acedida fora da *UI Thread*, visto que *Android UI toolkit* não é *Thread-safe*, por isso não devem ser manipulados a partir do *worker thread*. Todas as manipulações a interface de utilizadores (UI) devem ser realizadas no *UI Thread*[38].

A.4.1 Soluções

Uma boa solução para realizar operações em *worker Thread* e atualizar a interface do utilizador é a *AsyncTask*. Esta classe permite realizar operações que bloqueiam a *UI Thread* numa *worker Thread* e publicar os resultados na *UI Thread*, sem requerer a manipulação de *Threads* e outras técnicas mais complexas. Para utilizá-lo deve-se estender a essa classe e implementar os métodos *callback*:

- *doInBackground()* – este método corre as operações que se pretende a um conjunto de *threads* em *background*. Em qualquer momento no *doInBackground()* pode-se chamar o método *publishProgress()* de forma a ser executado o método *callback onProgressUpdate()*.
- *onPostExecute()* – este método entrega os resultados obtidos no *doInBackground()* e corre em *UI Thread*, permitindo a atualização da interface do utilizador de uma forma segura. Para além do *onPostExecute()*, são também invocados no *UI thread* os métodos *callback onPreExecute()* e *onProgressUpdate()*.

AsyncTask é uma solução ideal para realizar operações que demoram alguns segundos (como por exemplo descarregar uma foto, ficheiros, etc.), mas quando é preciso manter a *thread* a correr por um tempo longo indeterminado, e/ou implementar métodos a serem chamados por mais de

uma *Thread*, será necessário mecanismos mais complexos. Porém, nesta situação o *AsyncTask* não é recomendado [38].

A.4.2 Métodos Thread-safe

Para se ter uma aplicação *thread-safe*, deve-se também, em alguns casos (situações em que os métodos são chamados em mais de uma *thread*), escrever os métodos para serem *thread-safe*.

Uma boa forma para solucionar esse tipo de situação é a utilização de mecanismos disponíveis no *Android*, para comunicação entre processos (*interprocess communication - IPC*), utilizando chamadas de procedimentos remotos (*remote procedure calls - RPCs*), onde os métodos são chamados pela *Activity* ou outros componentes da aplicação, mas serão executados remotamente em outro processo, retornando o resultado de volta a onde foi chamado. Estes procedimentos envolvem a decomposição do método chamado e seus dados para um nível em que o sistema operativo possa entender, e transmiti-lo do processo e espaço de endereçamento local para o processo e espaço de endereçamento remoto, onde irá ser remontado e restabelecido a chamada. Todo o código da realização de transações IPC é fornecido pelo *Android*, de modo a permitir que o programador apenas foque na definição e implementação da interface de programação *RPC*[38].

A.5 AIDL - Android Interface Definition Language

O Android Interface Definition Language é um dos mecanismos que permite a realização de comunicação entre processos (IPC) no *Android OS*, tornando o seu uso mais indicado quando se pretende desenvolver um Serviço em *Android* que permite acesso a clientes de diferentes aplicações a este serviço para IPC, e/ou lidar com múltiplos *threads* dentro do serviço.

O *AIDL* permite ao programador definir interface de programação onde é formalizado um acordo entre o cliente e o servidor sobre a comunicação entre eles utilizando. Funciona de uma forma similar a qualquer outro interface Java, onde são definidos métodos abstratos. Para utilizá-lo é necessário criar um ficheiro com a extensão .aidl e definir todos os métodos abstratos que serão acedidos a partir de processos externos.

AIDL suporta transmitir entre processos (retornar/passar parâmetros) apenas tipo de dados primitivos e alguns tipo de dados básicos como *String*, *List* e *map*, caso se pretende utilizar outro tipo de dados (objecto) é necessário implementar a interface *Parcelable* na classe do objecto (tipo de dado) que se pretende transmitir, e criar o seus respectivo ficheiro .aidl, antes de o importar no AIDL para IPC.

A Implementação da interface *Parcelable* na classe do objeto que se pretende transmitir entre processos é fundamental, pois irá permitir ao sistema *Android* decompor o objeto em tipos primitivos de modo a ser *marshalled*(processo de transformação da representação memória de um objeto para um formato de dados apropriado para armazenamento ou transmissão) em todos

os processos [39].

Apêndice B

API Bluetooth

As *APIs Bluetooth* permitem aceder as funcionalidades do *framework Bluetooth* existente nos dispositivos *Android*. Através dessas *APIs* as aplicações conseguem usufruir de funcionalidades como: procurar por outros dispositivos *Bluetooth* se encontram no seu raio de alcance; pesquisar por dispositivos *Bluetooth* emparelhados; estabelecer canais *RFCOMM*; conectar-se a outros dispositivos através da descoberta de serviços; transferir e receber dados de e para outros dispositivos; manipular múltiplas conexões [1]. Todas as *APIs Bluetooth* encontram-se no pacote *android.bluetooth* do *SDK Android*. Na tabela B.1 encontram as classes que utilizamos na implementação do *middleware* de comunicação.

Uma aplicação *Android OS* para utilizar os recursos *Bluetooth*, precisa antes de mais, declarar pelo menos uma destas duas permissões: *BLUETOOTH* e *BLUETOOTH_ADMIN*. A requisição de permissão BLUETOOTH é necessário para qualquer tipo de comunicação via Bluetooth que uma aplicação deseja realizar, como por exemplo: requisitar conexão, aceitar uma conexão e transferir dados. Já a requisição da permissão *BLUETOOTH_ADMIN* é necessário para que a aplicação possa iniciar a descoberta de dispositivos ou manipular as configurações do *Bluetooth*. Tendo essas permissões, a aplicação pode utilizar BluetoothAdapter que representa o próprio adaptador *Bluetooth* do dispositivo, para usufruir dos recursos da tecnologia *Bluetooth* existente no dispositivo móvel, sendo que, para qualquer atividade no *framework Bluetooth* é necessário *BluetoothAdapter* [1].

B.1 Descoberta de dispositivos

Para que uma aplicação possa inicializar a descoberta de dispositivos, este terá que chamar o método *startDiscovery()* do *BluetoothAdapter*. O processo de descoberta é assíncrono, portanto *startDiscovery()* ao ser chamado retorna automaticamente, podendo a aplicação registrar as intents *ACTION_DISCOVERY_STARTED* e *ACTION_DISCOVERY_FINISHED* para receber através de *brodcastReceiver*, o momento exato que o processo de descoberta iniciou e o momento que terminou. Este processo de descoberta tem normalmente um tempo de duração de 12

Tabela B.1: Classes das APIs Bluetooth [1]

Classe	**Descrição**
BluetoothAdapter	Representa o Adaptador Bluetooth Local. Através de BluetoothAdapter é que uma aplicação consegue interagir com o Bluetooth,podendo realizar operações como: descobrir dispositivos Bluetooth, consultar uma lista de dispositivos emparelhados, instanciar um BluetoothDevice utilizando um endereço MAC conhecido, e criar um BluetoothServerSocket para ficar a escuta de conexões de outros dispositivos.
BluetoothDevice	Este representa um dispositivo Bluetooth remoto, é normalmente utilizado para consultar informações sobre um dispositivo (informações como nome, endereço, e estado da conexão), e para requerer uma conexão com um dispositivo remoto através de um BluetoothSocket.
BluetoothSocket	Representa a interface para um socket Bluetooth (semelhante a um Socket TCP). BluetoothSocket é um ponto de conexão que permite a uma aplicação trocar dados com um outro dispositivo Bluetooth através do InputStream e OutputStream.
BluetoothServerSocket	Este representa um socket servidor que mantém-se a escuta de requisições que chegam ao dispositivo a, qual a aplicação se encontra a correr (semelhante a um ServerSocket TCP). Para que dois dispositivos Android possam conectar entre si através de Bluetooth, um dos dispositivos precisa abrir um socket servidor com esta classe, e quando chegar no dispositivo uma requisição de conexão proveniente de dispositivos Bluetooth remotos, o BluetoothServerSocket irá retornar um BluetoothSocket conectado assim que aceitar a conexão.

segundos. É de referir, que é necessário registrar *ACTION_FOUND*, de forma a permitir que a aplicação seja notificada, a cada dispositivo *Bluetooth* remoto encontrado [1].

B.2 Permitir a descoberta do dispositivo

Para que um dispositivo seja descoberto por outros dispositivos, este terá que ser ativado para o fazer. Através do método *startActivityForResult(Intent, int)* da *BluetoothAdapter*, tendo

ACTION_REQUEST_DISCOVERABLE como ação da *Intent*, a aplicação consegue disparar uma requisição para ativar o modo que permite o dispositivo ser descoberto, sem ser preciso interromper a aplicação em questão. Por defeito o dispositivo mantem-se habilitado a ser descoberto por outros dispositivos durante 120 segundos, mas pode-se definir uma duração diferente através do acréscimo de um parâmetro extra (*EXTRA_DISCOVERABLE_DURATION*) ao *Intent* . A duração máxima que uma aplicação possa definir para manter um dispositivo apto a ser descoberto é 3600 segundos, e o valor 0 significa que o dispositivo irá manter-se sempre disponível a ser descoberto por outros dispositivos [1].

B.3 A conexão entre dois dispositivos através de uma aplicação

both the server-side and client-side mechanisms Para que uma aplicação realize conexão entre dois dispositivo, é preciso implementar o mecanismo para estabelecer conexão tanto do lado do servidor como do cliente, uma vez que, um dispositivo precisa abrir um servidor e ficar à espera de conexão, e o outro precisa iniciar a conexão, utilizando o endereço MAC do servidor. É considerado que um servidor e um cliente estão conectados entre si, quando tiverem os dois um *BluetoothSocket* conectado no mesmo canal *RFCOMM*. O *BluetoothSocket* conectado é recebido de forma diferente nos dois lados, o servidor irá recebê-lo quando aceitar uma conexão, e cliente irá recebê-lo quando abrir um canal *RFCOMM* para o servidor. Cada um dos dispositivos podem obter *InputStream* e *OutputStream* para realizarem a transferência de dados entre si [1].

Quando a conexão entre dois dispositivo é estabelecida pela primeira vez, é apresentado automaticamente ao utilizador um pedido de emparelhamento, podendo o utilizador aceitar ou não. Dois dispositivos ao serem emparelhados, guardam as informações básicas (como nome do dispositivo, classe, endereço MAC) um do outro, sendo que essas informações podem ser lidas utilizando as *APIs Bluetooth*. Tendo o endereço MAC do dispositivo remoto já conhecido, pode-se iniciar uma conexão com esse dispositivo a qualquer momento, não carecendo da realização de descoberta de dispositivo, isto assumindo que o dispositivo esteja ao alcance.

É de se salientar que estar emparelhado é diferente de estar conectado. Estar emparelhado significa que os dois dispositivos estão cientes da existência um do outro, possuem uma chave de ligação compartilhada que pode ser usada para autenticação, e são capazes de estabelecer uma conexão encriptada entre eles. Estar conectado significa que os dispositivos no momento compartilham uma canal *RFCOMM* e são capazes de transmitir dados entre eles. As *APIs Bluetooth* do *Android* atual, requer que os dispositivos sejam emparelhados antes que uma conexão *RFCOMM* possa ser estabelecida. O emparelhamento é o primeiro a ser executado e acontece de forma automática, sempre que é iniciado uma conexão criptografada por meio das *API Bluetooth*.

B.3.1 Conectar como servidor

Para que uma aplicação conecte o seu dispositivo como servidor (*Master*), este terá que obter *BluetoothServerSocket*, através da chamada do método *listenUsingRfcommWithServiceRecord(String, UUID)* do *BluetoothAdapter*. O parâmetro *string* do método é um nome que identifica o serviço, a qual o sistema irá escrever automaticamente como uma nova entrada do SDP (*Service Discovery Protocol*) do dispositivo, sendo que o nome é arbitrário, podendo simplesmente ser o nome de sua aplicação. E o parâmetro UUID (*Universally Unique Identifier*) é um formato padronizado para ID composto por *string* de 128 bits utilizado para identificar de forma única uma informação. O UUID é também incluído juntamente com *string* na entrada SDP, isto é quando um cliente deseja conectar a um *BluetoothServerSocket*, este terá que conter um UUID que identifica unicamente o serviço com o qual deseja se conectar, desta forma a conexão é aceite caso o UUID do cliente e do servidor coincidirem-se [1].

Uma aplicação que já contém um *BluetoothServerSocket*, pode iniciar o processo de espera por requisição de conexão do *BluetoothServerSocket* através da chamada do método *accept()*. Esta chamada é bloqueante, portanto irá bloquear a *Thread* onde se encontra a executar, até ser aceite uma conexão, sendo que, uma conexão só é aceite quando o dispositivo remoto envia uma solicitação de conexão com um UUID que coincide com o que se encontra registrado no *BluetoothServerSocket*. Quando a conexão for bem sucedida, o *accept()* irá retornar um *BluetoothSocket* conectado.

B.3.2 Conectar como cliente

Para iniciar uma conexão com um dispositivo remoto, a aplicação terá que obter o objecto *BluetoothDevice* que representa o dispositivo remoto, normalmente obtido através do processo de descoberta de dispositivos. Tendo o *BluetoothDevice* remoto que se pretende conectar, é preciso obter um *BluetoothSocket* através do método *createRfcommSocketToServiceRecord(UUID)* do *BluetoothDevice*. Tendo *BluetoothSocket* pode-se iniciar a conexão através da chamada ao método *connect()*, com essa chamada o sistema irá executar uma busca SDP no dispositivo remoto para verificar se o UUID coincide, caso a busca for bem sucedida e o dispositivo remoto aceitar a conexão, este irá compartilhar o seu canal *RFCOMM* para ser utilizado durante a conexão. O método *connect()* é bloqueante, só retorna após executar todo o seu processo. Se por qualquer razão, a conexão falhar ou o método *connect()* demorar mais de 12 segundo, é disparada uma exceção [1].

Apêndice C

APIs Wi-Fi Direct

As *APIs Wi-Fi Direct* foram introduzidas no *Android SDK* a partir da versão *Android 4.0* em dispositivos contendo hardware apropriado. *APIs Wi-Fi Direct* permitem as suas aplicações interagirem com o *framework Wi-Fi Direct*, de modo a poderem usufruir dos recursos fornecidos por este *framework*, permitindo assim que a aplicação execute diversas operações, tais como: descobrir dispositivos *Wi-Fi Direct* próximos, conectar a esses dispositivos e comunicar com eles.

A tabela C.1 apresenta a lista de classes que fazem parte das *APIs Wi-Fi Direct*, bem como a sua descrição. As *APIs Wi-Fi Direct* encontram-se constituídas pelas seguintes partes principais [2]:

- Métodos que permitem realizar descobertas, pedidos e conexão de pares. Estes métodos encontram-se definidas na classe *WifiP2pManager*.
- *Listeners* que permitem notificar a aplicação sobre sucesso ou falha dos métodos de *WifiP2pManager* chamados. Quando é efetuado uma chamada aos métodos de *WifiP2pManager*, esses métodos podem receber como parâmetro um *listener* especifico.
- *Intents* permite notificar a aplicação sobre eventos específicos que ocorrem na *framework Wi-Fi Direct*, como a perda de uma conexão ou um par recém-descoberto

A tabela C.2 descreve os métodos fornecidos pela classe *WifiP2pManager* que permitem a interação com o *framawork Wi-Fi Direct* do dispositivo.

A tabela C.3 contém a lista de *listeners* e os respectivos métodos da classe *WifiP2pManager*, onde podem ser passados como parâmetros, para que o *framework Wi-Fi Direct* possa notificar as aplicações sobre o estado da chamada do método realizado. Na tabela C.4 encontram a descrição desses *listeners*.

Por último a tabela C.5 contém a lista de *intents* que realizam *broadcast* quando acontece um dado evento no *Wi-Fi Direct*. Para que uma aplicação receba essas notificações, é preciso que registe nele esses *intents*, através da criação de *broadcast receiver* que irá lidar com os *intents*.

Tabela C.1: As classes que fazem parte das *APIs Wi-Fi Direct* [2]

Class	Descrição
WifiP2pManager	A classe WifiP2pManager é o adaptador da interface Wi-Fi Direct local. Nesta classe que é fornecido a API para manipular conectividades da Wi-Fi Direct, permitindo a aplicação descobrir pares, estabelecer conexões e pesquisar por lista de pares
Channel	É a classe que liga a aplicação ao framework Wi-Fi Direct. A maioria das operações requerem Channel como argumento. Uma instância deste objeto é obtido através da chamada ao initialize(Context, Looper, WifiP2pManager.ChannelListener)
WifiP2pInfo	A classe representa a informação de conexão sobre o P2P Group
WifiP2pGroup	Esta classe representa um P2P Group.
WifiP2pDevice	Esta classe representa um P2P Device
WifiP2pDeviceList	Esta classe representa uma lista de P2P Device
WifiP2pConfig	Esta classe representa a configuração de Wi-Fi Direct para estabelecer uma conexão
WpsInfo	Esta classe representa Wi-Fi,Protected Setup

Tabela C.2: Métodos do Wi-Fi P2P [2]

Método	Descrição
Initialize()	Este é o primeiro método a ser chamado numa aplicação, para a utilização da tecnologia Wi-Fi Direct dos dispositivos Android. Através desse método que uma aplicação irá registrar ao framework Wi-Fi Direct.
connect()	Inicializa uma conexão peer-to-peer,com um dispositivo, com a configuração especificada
cancelConnect()	Cancela qualquer negociação de P2P Group em andamento
requestConnectInfo()	Realiza pedido de informação da conexão do dispositivo
createGroup()	Cria um P2P Group autonomous, onde o dispositivo irá si auto nomear GO
removeGroup()	Remove o P2P Group actual
requestGroupInfo()	Realiza,o pedido de informação do P2P Group
discoverPeers()	Inicializa a descoberta de pares vizinhos
requestPeers()	Realiza o pedido a lista actual de pares decobertos

C.1 Criar aplicações que utilizam Wi-Fi Direct

Antes de se utilizar *APIs Wi-Fi Direct*, é preciso garantir que a aplicação terá acesso ao hardware, e que o dispositivo suporta os protocolos *Wi-Fi Direct*. Caso *Wi-Fi Direct* seja suportado,

Tabela C.3: Listeners Wi-Fi Direct e os seus métodos associados [2]

Interface Listener	**Ações associados**
WifiP2pManager.ActionListener	connect(),cancelConnect(), createGroup(), removeGroup(), e discoverPeers()
WifiP2pManager.ChannelListener	initialize()
WifiP2pManager.ConnectionInfoListener	requestConnectInfo()
WifiP2pManager.GroupInfoListener	requestGroupInfo()
WifiP2pManager.PeerListListener	requestPeers()

Tabela C.4: Listeners Wi-Fi Direct [2]

Listener	**Descrição**
WifiP2pManager.ActionListener	Interface para invocar callback em uma ação da aplicação
WifiP2pManager.ChannelListener	Interface para invocar callback quando for perdido o canal da framework
WifiP2pManager.ConnectionInfoListener	Interface para invocar callback quando estiver disponível, informações de conexão
WifiP2pManager.GroupInfoListener	Interface para invocar callbackquando estiver disponível, informações de P2P Group
WifiP2pManager.PeerListListener	Interface para invocar callback quando estiver disponível, lista de pares

Tabela C.5: Intents do Wi-Fi P2P

Intent	**Descrição**
WIFI_P2P_CONNECTION_CHANGED_ACTION	Broadcast quando o estado da conexão do dispositivo Wi-Fi alterar
WIFI_P2P_PEERS_CHANGED_ACTION	Broadcast quando é chamado o método discoverPeers(), que normalmente é chamado o método requestPeers() para ser recebido na aplicação, a lista de pares encontrados
WIFI_P2P_STATE_CHANGED_ACTION	Broadcast quando Wi-Fi Direct é inicializado ou desligado no dispositivo
WIFI_P2P_THIS_DEVICE_CHANGED_ACTION	Broadcast quando acontecer alguma alteração nos detalhes do dispositivo, como por exemplo alteração do nome do dispositivo

pode-se obter uma instância do *WifiP2pManager* que irá ser utilizado para registar a aplicação ao *framework Wi-Fi Direct*, através da chamada do método *initialize()*. Este método retorna um *WifiP2pManager.Channel*, que será utilizado para ligar aplicação ao *framework Wi-Fi Direct*.

Tendo esses dois objetos *WifiP2pManager* e *WifiP2pManager.Channel*, as aplicações já podem chamar os métodos da classe *WifiP2pManager* para utilizarem as funcionalidades fornecidas pelo *framework Wi-Fi Direct*. Além disso é preciso criar *broadcast receiver*, no caso da aplicação pretender receber notificações de eventos que lhe interessam, como por exemplo: saber se o *Wi-Fi* está ligado ou não, se o dispositivo perdeu a conexão a uma rede ou conectou-se a uma rede, etc [2].

C.1.1 Conexão Wi-Fi Direct

Utilizando as *APIs Wi-Fi Direct* do *SDK Android*, existe duas formas de se conectar um dispositivo a um *P2P Group*. Estes são realizadas através da chamada aos métodos *connect()* e *createGroup()* da classe *WifiP2pManager* [2].

No método *createGroup()*, será realizado a técnica de formação de grupo *Autonomous*, portanto o dispositivo irá criar um *P2P Group* onde ele é *P2P GO*. E através do método *connect()* um dispositivo irá conectar-se a um outro dispositivo previamente descoberto no processo de descoberta de pares. Para chamar o método *connect()* é necessário passar como parâmetro o objecto *WifiP2pConfig* que contém a configuração para estabelecer a conexão com o dispositivo desejado.

Na conexão através da chamada ao método *connect()*, os dispositivos irão formar um *P2P Group*, negociando entre si qual deles é que irá realizar o papel de *P2P GO*, (isto acontece no caso de nenhum dos dispositivos fazerem parte de um *P2P Group*) caso o dispositivo que irá realizar *connect()*, já se encontra num *P2P Group* e este for *P2P GO*, a função *connect()* efetua um convite ao dispositivo que este deseja conectar, para se juntar ao grupo. Um dispositivo que não se encontra num *P2P Group*, pode utilizar a função *conenct()* com o objectivo de se conectar a um dispositivo *P2P GO*, juntando assim ao seu *P2P Group*, Sendo irrefutável que o dispositivo terá que descobrir o dispositivo *P2P GO* antes de executar a função *connect()*.

C.1.2 Descoberta de Pares

Segundo a especificação *Wi-Fi Direct*, para realizar uma conexão com um outro dispositivo *Wi-Fi Direct*, é preciso que este seja anteriormente descoberto. Nas *APIs Wi-Fi Direct* do *sdk Android*, a descoberta de pares pode ser realizada através da chamada do método *discoverPeers()* da classe *WifiP2pManager*. No entanto, esta não é uma boa forma para se descobrir com que par vizinho este deve conectar. Pois, o *discoverPeers()* realiza a descoberta de todos os pares *Wi-Fi Direct* no seu raio de alcance, não levando em consideração as características dos pares. A título de exemplo, é apresentado o seguinte cenário: Uma aplicação de jogos cujo objectivo é descobrir os nós vizinhos que estão a correr o mesmo. Se a aplicação utilizar *discoverPeers()* não terá como descobrir quais são os nós vizinhos que estão a correr o mesmo jogo, se não existir uma conexão previamente estalecida a estes nós. Desta forma, para solucionar este tipo de cenário foi lançado posteriormente na versão *Android 4.1*, *APIs* adicionais as *APIs Wi-Fi Direct*,

que permitem realizar a descoberta de serviços nos dispositivos *Wi-Fi Direct* mesmo antes de realizarem conexões IP entre eles [2].

C.1.2.1 Descoberta de Serviço

A versão *SDK Android 4.1* e as mais recentes, incorporaram a oportunidade das aplicações pre associarem a descoberta de serviço, permitindo assim filtrar os pares com base nos serviços que se encontram em execução. Desta forma as aplicações poderão anunciar um serviço para uma aplicação em um dispositivo vizinho, mesmo antes de realizarem conexão IP entre eles. O *DNS based service discovery (Bonjour)* e o UpnP são os protocolos para descoberta de serviço suportados atualmente pelo *framework Wifi Direct*. Os recursos do *Bonjour* podem ser encontrados em *dns-sd.org*, e os de UpnP em *upnp.org*.

Para que seja possível a descoberta de serviços nos dispositivos *Wi-Fi Direct*, foram incorporados a partir da versão *SDK Android 4.1*, métodos e interfaces (*Listeners*) adicionais a classe *WifiP2pManager* assim como, novas classes as *APIs Wifi Direct*, de forma a permitir que os dispositivos *Wi-Fi Direct* pre associem descoberta de serviço com finalidade de descobrir pares contendo os serviços desejados.

A tabela C.7 contém os métodos (e suas respectivas descrições) que foram adicionados à classe *WifiP2pManager*, a fim de permitir a manipulação de descoberta de serviços na *framework Wifi Direct*. Na tabela C.6, encontram as interfaces (*listeners*) para invocar *callback* quando o *framework Wifi Direct* receber respostas de descoberta de serviços e os dados *TXT record* transportados. A tabela C.8 encontram as classes que permitem criar esses serviços [2].

Tabela C.6: *Listeners* adicionados as *APIs Wi-Fi Direct* para realizar descoberta de Serviço

Listener	**Descrição**
WifiP2pManager.DnsSdServiceResponseListener	Interface para invocar callback quando for recebido resposta de descoberta de serviço Bonjour
WifiP2pManager.DnsSdTxtRecordListener	Interface para invocar callback quando TXT record de Bonjour estiver disponível para um serviço
WifiP2pManager.UpnpServiceResponseListener	Interface para invocar callback quando for recebido resposta de descoberta de serviço Upnp

Tabela C.7: Métodos adicionados a *WifiP2pManager* para descoberta de Serviço

Método	**Descrição**
addLocalService()	Registra um serviço local para a descoberta de serviço
addServiceRequest()	Adiciona um pedido de descoberta de serviço
clearLocalServices()	Remove todos os serviços locais registrados para descoberta de serviço
clearServiceRequests()	Remove todos os pedidos de descoberta de serviço registrados
removeLocalService()	Remove o serviço local registado através de addLocalService(WifiP2pManager.Channel, WifiP2pServiceInfo, WifiP2pManager.ActionListener)
removeServiceRequest()	Remove o pedido de descoberta de serviço adicionado através deregistado através de addServiceRequest(WifiP2pManager.Channel, WifiP2pServiceRequest, WifiP2pManager.ActionListener)
discoverServices()	Inicializa a descoberta de serviço. O processo de descoberta envolve a verificação de pedidos de serviços, com o objetivo de se estabelecer conexão com pares que suportam um serviço desejado.Para descobrir um serviço é preciso especifica-lo através da chamada ao método addServiceRequest(WifiP2pManager.Channel, WifiP2pServiceRequest, WifiP2pManager.ActionListener)
setDnsSdResponseListeners()	Regista o callback a ser invocado ao ser recebido resposta de descoberta de serviço Bonjour
setServiceResponseListener()	Regista o callback a ser invocado ao ser recebido resposta de descoberta de serviço
setUpnpServiceResponseListener()	Regista o callback a ser invocado ao ser recebido resposta de descoberta de serviço Upnp

C.2 Bonjour

O *Bonjour* é um protocolo para descoberta de serviços, tem os seus recursos definidos em DNS-SD [40] que especifica como nomear e como estruturar *DNS records* para facilitar a descoberta de serviços. Este mecanismo permite ao Cliente descobrir a lista dos nomes de instância de serviços

Tabela C.8: Classes adicionais a *APIs Wi-Fi Direct* para descoberta de Serviço

Class	**Descrição**
WifiP2pServiceRequest	Esta é a classe para criar pedido de descobertade serviço para ser utilizado com addServiceRequest(WifiP2pManager.Channel, WifiP2pServiceRequest, WifiP2pManager.ActionListener) e removeServiceRequest(WifiP2pManager.Channel, WifiP2pServiceRequest, WifiP2pManager.ActionListener).
WifiP2pUpnpServiceRequest	Esta é a classe para criar pedido de descoberta de serviço Upnp para ser utilizado com addServiceRequest(WifiP2pManager.Channel, WifiP2pServiceRequest, WifiP2pManager.ActionListener) e removeServiceRequest(WifiP2pManager.Channel, WifiP2pServiceRequest, WifiP2pManager.ActionListener)
WifiP2pDnsSdServiceRequest	Esta é a classe para criar pedido de descoberta de serviço Bonjour para ser utilizado com addServiceRequest(WifiP2pManager.Channel, WifiP2pServiceRequest, WifiP2pManager.ActionListener) e removeServiceRequest(WifiP2pManager.Channel, WifiP2pServiceRequest, WifiP2pManager.ActionListener)
WifiP2pServiceInfo	É a classe para guardar a informação de serviço que é anunciado através da configuração,do Wi-Fi Direct
WifiP2pDnsSdServiceInfo	É a classe para guardar a informação deserviço Bonjour que é anunciado através da configuração,do Wi-Fi Direct
WifiP2pUpnpServiceInfo	É a classe para guardar a informação de serviço Upnp que é anunciado através da configuração,do Wi-Fi Direct

que ele deseja. DNS-SD especifica que uma instância de um serviço pode ser descrito utilizando *DNS SRV record* e *DNS TXT records* [40].

DNS SRV record descreve o serviço com a seguinte estrutura:
"*Instance.Service.Domain*" juntamente com a porta em que o serviço está a correr [40].

DNS TXT record, fornece informações adicionais sobre a estrutura de serviços como series de pares *key/value*. Cada *string* que constituem um *DNS TXT record* tem um limite de 255 *bytes*, e encontram-se no formato "*key=value*".

O "Service" da estrutura de *DNS SRV record*, apresenta o seguinte formato:
"nome do protocolo de aplicação, tipo serviço"."tcp ou udp, ou seja protocolo de transporte". Um exemplo do formato de "Service": _http._tcp

Um cliente descobre a lista de instâncias disponíveis de um determinado tipo de serviço, utilizando uma consulta para *DNS PTR record*, com o nome no formato "*serviço.Domain*" A consulta irá retornar um conjunto de zero ou mais nomes, que são os nomes dos pares *DNS*

SRV/TXT record acima mencionados [40].

Apêndice D

Acrónimos

ACL Asynchronous Connectionless

AIDL Android Interface Definition Language

AP Access Point

API Application Programming Interface

ANQP Access Network Query Protocol

CCK Complementary Code Keying

D2D Device-to-Device

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

DNS-SD DNS Service Discovery

FHSS Frequncy Hopping Spread Spectrum

GAS Generic Advertisement Protocol

GO Group Owner

IDE Integrated Development Enviroment

IP Internet Protocol

IPC Inter-Process Communication

ISM Industrial Scientific and Medical

JVM Java Virtual Machine

LMP Link Manager Protocol

MAC Media Access Control

MIMO Multiple-Input-Multiple-Output

OFDM Orthogonal Frequency Division Multiplexing

OS Operating System

OSI Open Systems Interconnection

P2P peer-to-peer

QoS Quality of Service

RF Radio Frequency

RPC Remote Procedure call

RO-WDB Rede Oportunístico Wi-Fi Direct Bluetooth

SCO Synchronous Connection Oriented

SDK Software Development Kit

SDP Service Discovery Protocol

SIG Special Interest Group

SIP Session Initiation Protocol

TCP Transmission Control Protocol

TDD Time Division Duplex

UDP User Datagram Protocol

UI User Interface

UUID Universally Unique Identifier

VoIP Voice over Internet Protocol

WLAN Wireless Local Area Network

WPAN Wireless Personal Area Network